UC Berkeley
Research Reports

Title
A Specification Of An Automated Freeway With Vehicle-borne Intelligence

Permalink
https://escholarship.org/uc/item/67b3b8zv

Author
Hitchcock, Anthony

Publication Date
1992

eScholarship.org Powered by the California Digital Library
University of California

This paper has been mechanically scanned. Some errors may have been inadvertently introduced.

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

A Specification of an Automated Freeway with Vehicle-Borne Intelligence

Anthony Hitchcock

UCB-ITS-PRR-92-18

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration and National Highway Traffic Safety Administration.

The contents of this report reflect the views of the author who is responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the State of California. This report does not constitute a standard, specification, or regulation.

DECEMBER 1992

ISSN 1055-1425

CONTENTS

Appendixes

List of Figures

GLOSSARY OF TERMS

In this paper, a number of specially-defined terms of art are used, which include abbreviations. The following table refers to the definitions of the terms and spells out the abbreviations.

Term	*Section*
AL = automated lane	1
AL license	4.1
AR = asynchronous record	*4.4*
b.e.m. = block entry marker	*4.5*
block	*4.5*
C mode = Crashstop	*6*
CA mode = Closed-Ahead	*6*
dormitory	*4.1*
fence	*5*
follower	*2*
free agent	*2*
full platoon braking	*2*
gate	*5*
loc = location = odometer reading and lane #	*4.5*
manual spacing	*2*
MON = self-monitor	*4.5*
N mode = Normal	*6*
NE mode = No-Entry	*6*
platoon leader	*2*
platoon spacing	*2*
R mode = Resume	*6*
S mode = Stop	*6*
SA mode = Slow-Ahead	*6*
TL = Transition lane	*4.6*
turning-point	*4.6*
VPD = Vehicle Presence Detector	*4.6*
v s v = Vehicle-borne State Vector	*4.4*

A Specification of an Automated Freeway with Vehicle-Borne Intelligence

1. INTRODUCTION

This paper is complete in itself. However, along with another paper (Hitchcock, 1992b) it can also be regarded as an appendix to "Methods for Analysis of IVHS Safety: Final Report of PATH MOU 19" (Hitchcock, 1992a). Readers not familiar with the area are strongly advised to read the other report first. An even shorter account of the background can be found in Hitchcock 1991.

Further, the work here extends the work of Hsu, et al. (1991). The work plan of the whole study included two worked examples of automated freeways. The overall objective was to demonstrate and refine the techniques of specification and safety analysis. The two examples should therefore be very different. The first was a single-AL system (AL = automated lane). Intelligence was concentrated in the infrastructure. It was therefore natural to look at a multi-AL system with mainly vehicle-borne intelligence.

Hsu and her colleagues had quite different concerns. Their purpose was to investigate a method of proving conformity of a conceptual design to its specification. This method used the logical technique enshrined in a computer program called COSPAN. The example they used to make their tests concrete was a partial design of an automated freeway. The freeway had many lanes. Intelligence was entirely vehicle-borne in the lower levels of the architectural hierarchy (see Section 4). This work was also part of the PATH program. It seemed sensible to build on Hsu's design in the present work. This was particularly the case since Hsu and her colleagues were able to show that their design did meet their partial specification. Accordingly, the permission of the authors was sought. The design proposed by Hsu, et al. (1991) is close to being a subsystem of the design set out here. However, Hsu's design is incomplete for the present purpose. We have extended it very considerably. Some changes have also been made to what Hsu, et al. (1991) proposed.

The specification for an automated freeway is set out in a fully formal manner in this paper. Subsequently, a series of safety analyses was carried out on the specification. These are reported elsewhere (Hitchcock, 1992b). The objective of the programme of work, of which this is part, is to derive a technique of safety analysis for such systems. The system reported here is the second example on which a trial analysis has been demonstrated. The analysis depends on the precise nature of the system specified. It is therefore necessary that the specification be recorded without possibility of ambiguity. Great detail is thus required. Also, the method of analysis recommended by Hitchcock (1992a) does require formal documentation. This paper is consequently intended as an exemplar of such documentation. This applies especially to the appendixes. In the appendixes a formalized language has been used, which is analogous to some computer languages.

This report specifies the second example. To demonstrate the method of safety analysis a procedure called fault tree analysis was then applied. The fault tree for the present example is discussed in Hitchcock, 1992b.

The freeway specified here operates with vehicles in ***platoons.*** This is the basis on which other work in PATH has been carried out. There was no good reason to do other here. The safety argument in favour of platoons is reviewed in Hitchcock, 1992a.

The method of demonstrating safety used in the larger programme starts by defining certain ***hazards.*** A safety criterion is selected. The criterion used here is that two or more simultaneous faults must occur independently before the hazards can arise. The process is described in Hitchcock, 1991. Hazards are detailed later (see Section 3).

In what follows, we describe the whole design. Later, a short section (see Section 9) distinguishes the part that is due to Hsu and her colleagues from what is new.

2. SOME DEFINITIONS

Each vehicle will have a maximum deceleration when brakes are applied, which will vary between vehicles. Under some circumstances it will be necessary to decelerate as quickly as possible, short of generating within-platoon collisions. The appropriate deceleration is called ***full platoon braking.*** It is clearly a function of the road surface condition, and is therefore set by each block as vehicles enter.

Platoon spacing is that spacing within which a vehicle decelerating at full platoon braking can avoid colliding with one ahead when the latter is decelerating at some standard greater rate (one might choose 0.8 g — a practical maximum). The following platoon is supposed to be warned within some standard interval.

Manual spacing is the spacing at which drivers normally drive without being alarmed. It is a function of vehicle speed and road surface condition. A parameter to describe the latter is passed by the block controllers on entry.

On ALs, vehicles move in platoons. The first is the ***platoon leader;*** others are followers. A one-vehicle platoon is called a ***free agent.***

3. H A Z A R D S

A hazard is defined as a situation in which if one further fault occurs, a high speed collision may ensue. The safety criterion as explained in Hitchcock, 1992a is that a hazard will not arise without two simultaneous faults of system components. What we have to guard against are crashes arising from the way the system is designed. Not all accidents are of this kind. If, in particular, a driver causes his/her vehicle to collide with an automated one in a place where the presence of both is permitted, this is not a hazard.

High-speed collisions may occur:

a. when all platoons involved are under control, automated or manual (a free agent is a one-member platoon). In this case vehicles were too close to each other before a final control failure.

b. when one platoon is not under control. This will happen if automatic control is switched off before the driver is ready, or not switched on when the driver lets go.

In case a above, we are only concerned when the rear platoon is under automatic control. In this case (remembering that a free agent is a one-member platoon), a further failure will cause a catastrophe if:

Hazard 1. A platoon is separated from one ahead of it, or from a stationary object in its path, by less than platoon spacing; or

Hazard 2. A vehicle, not under system control, is at an unmeasured or unknown distance in front of a platoon.

Case b above refers to the case where neither driver nor system is in fact controlling the vehicle, or where a driver is placed in a situation where he/she cannot control it.

Hazard 3. **A** vehicle is released to manual control before the driver has given a positive indication that he/she accepts it.

Hazard 4. A vehicle is released to manual control at less than manual spacing from the vehicle ahead of it; or at such a relative speed that manual spacing will be realized in less than 2 seconds; or while the brakes are being applied.

Danger may arise if a driver tries to surrender control before the system accepts it. This is an aberration of a manual driver, but, by definition, it does not give rise to hazards.

4. SYSTEM ARCHITECTURE

An AVCS system architecture of a system similar to this one has been described by Varaiya and Shladover (1991). This architecture is more general than the authors claim, and we shall use Varaiya and Shladover's language to describe the architecture of the present system. The architecture is illustrated in Figure 1. It is hierarchical. Also, it is composed of modules. Modules may be hardware or software. The modules communicate with others at the same level in the hierarchy, and also with modules one level up and one level down. Their interfaces are fully specified. The significance of this is that if an improved module design is subsequently produced it can be slotted into the system without redesigning of the whole. In particular, it becomes possible to replace a single non-safety critical module with impunity. A total reappraisal of safety is not required.

The lower levels of the hierarchy are responsible for control of vehicle movement. They are safety-critical, as Figure 1 indicates. The highest level (this is an addition to the Varaiya and Shladover description) is law.

4.1 Architecture, Level 5: Law

The relevant law will control operations of automated freeways in many cities. Many of the problems which will arise are not covered by the present law. Here we shall state only the law, common to all city systems, which is required for operation. There are also a variety of wider areas where legal or quasi-legal regulation will be required. These include the mechanism for setting standards and ensuring or certifying conformity with them. We assume that the law will say:

a. Every vehicle trying to enter the AL will bear a remotely readable record which certifies that it is equipped with equipment of appropriate design. There will be a validation bit which indicates that the equipment has not been diagnosed as faulty, either by the vehicle itself or by the system. This bit, together with some data about the vehicle, will be called the AL, *license.* Only vehicles with a valid AL license may enter or remain on the ALs. Only a suitably qualified person may revalidate an invalid AL license. If a license becomes invalid while on the Als (i.e., if the vehicle develops a fault) the vehicle should quit the ALs at once.

b. It will be an offence to fail to resume manual control on exit from the ALs. If a request to do so is disobeyed, the vehicle may be re-admitted to the system and carried to some point where it is safe to leave it. (Periodically there will be areas where such vehicles can be parked. We call them *dormitories.)*

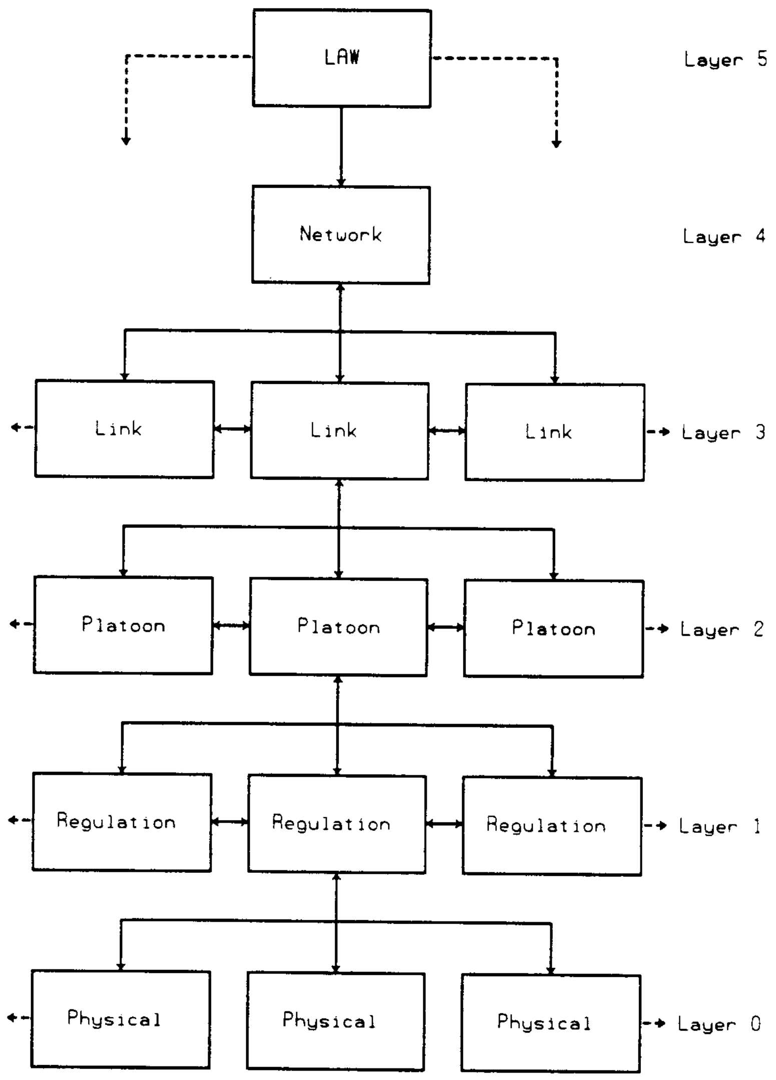


Figure 1. IVHS Control Architecture (after Varaiya and Shladover).

c. It will be an offencè to enter the system carrying an external or ill-secured load, or with a trailer not itself equipped with communication devices.

d. "Hacking," that is, emitting signals designed to influence the system to do something which it would not normally do, is illegal. This applies particularly to manually-controlled vehicles which declare themselves to have valid licenses.

4.2 Architecture, Level 4: Network

This controls more complex aspects of routing in the light of overall flows on the network. It also sets the parameters which correct the values of platoon spacing, etc., in the light of prevailing weather conditions. Physically, the network controller is infrastructure based. It will be connected to the link-level controllers, probably by hard-wiring.

4.3 Architecture, Level 3: Link

Link level prescribes the route to be followed by a vehicle through the system. The desired route is stored in the vehicle. The choice is informed both by the data from the network level and the destination stated by the vehicle on entry. When the time comes to change lanes, the lower levels read the internal record, and determine if local conditions permit the manoeuvre. If they do not, the change is made later. Under some conditions (e.g., lane closures) route may be updated after entry. The efficiency of the link level algorithms clearly affects the capacity of the ALs. Efficiency here is thus of economic importance.

Link level controllers are infrastructure-based. There will be one for every 5-20 miles of freeway. They can communicate directly (presumably at high or optical frequencies) with platoon-level controllers on vehicles in their area, and also with the platoon-level infrastructure-based controllers. This latter link is probably accomplished by hard-wiring. The platoon-level controllers at the roadside are effective only in fault conditions. Data are exchanged about lane closures, emergency operations, and the like. It is at this level, for example, that human supervisors or the Highway Patrol will intervene if it is necessary to manoeuvre an emergency vehicle on to the ALs.

Link level does need to know which vehicles are in its area and where they are going. However, this is not a safety-critical function.

4.4 Architecture, Level 2: Platoon

Platoon level is responsible for the manoeuvres which lead to entry, exit, changes of lane, and the formation and dissipation of platoons. Most of it is vehicle-borne. There is a platoon-level controller on each vehicle which oversees the actions of lower-level controllers of vehicle motion in such a way as to execute platoon manoeuvres safely and successfully at the right time and place. We will describe its operation later (see Section 8.3).

There is also a vehicle state vector *(VSV)*. The VSV is a software record containing the AL license, the route and a number of variables describing the current position of the vehicle and what it is doing. The VSV is held in an asynchronous record (AR). An AR is a computer storage device which can be written to and read from by several different computers which are not in sync. The VSV, in fact, can be accessed by all the controllers on the vehicle. It can also be accessed, via the communication devices, by other vehicles and the various roadside controllers. Among other fields, it contains a "busy" flag. This is set during any manoeuvre, and will usually preclude any other manoeuvre.

The fields in the VSV are these below:

ID#	vehicle identity
ln#	lane number
optsize	platoon optimum size
pltn#	platoon number
pos	position in platoon
busy	busy marker (see above)
sectlen	section length
fault	identity of any fault (see below)
mode	operating mode (see below) of current lane
xspeed	optimum speed in lane to right
hwy#	highway identity
sect#	section number
optspeed	platoon optimum speed
ownsize	size of own platoon
loc	odometer reading (see Section 4.5)
validation	valid AL license
speed	actual speed of vehicle
maxspd	maximum speed on freeway
safsp	minimum platoon separation at maxspeed

The state variables ***maxspd*** and ***safsp*** are set by the system on entry and are system-wide constants. They may vary from time to time, with the weather. There is a formula contained inside every vehicle which enables the safe interplatoon spacing to be calculated from these variables at any speed. *Optsize* is two numbers specifying a range of sizes within which platoons will neither merge or split. Like ***optspeed,*** it is set on entry by link and updated on each lane change.

At the roadside, there is a platoon-level controller associated with each block of freeway. A ***block*** is a length of some l-5 miles of freeway. It may be convenient to have one entrance and one exit point per block: the point is of no great importance. There is a platoon-level controller at each point where a lane-change can occur. These controllers are hard-wired to the other platoon-level controllers in the same block. They can also communicate with vehicles in their immediate neighbourhood. Their function is more fully discussed later (see Section 8.5).

4.5 Architecture, Level 1: Regulation

The regulatory layer controls the movements of individual vehicles in response to signals from the platoon level and sensor readings from the physical level. It is entirely vehicle-borne. There are many components:

a. ***Longitudinal Control*** System. This will maintain a vehicle in platoon. If the vehicle is a platoon leader or a free agent, the control system will maintain the vehicle at platoon spacing from the next vehicle ahead in the same lane. Alternatively, the system may be falling back from the vehicle ahead or joining up to it when some special manoeuvre is being carried out.

b. ***Lateral Control System.*** This will maintain a vehicle on track in lane. It will also change lane at appropriate locations when requested to do so.

c. ***Odometer.*** As will be seen, at the entry to every block there is a block entry marker ***(b.e.m.). The*** odometer measures the distance of the vehicle from the b.e.m. and records this number in the VSV. The lane # is also recorded there. We call this data Zoc, the vehicle’s location. On entry, a loc is input by the system.

d. ***Forward Sensor.*** This instrument measures the distance of the vehicle from the one ahead. It also measures relative velocity directly. The readings are stored in an AR. It will operate at close range, in platoon. At longer ranges, it can keep track of which of several vehicles ahead is in the same lane. It will always detect vehicles up to a range exceeding the maximum value of platoon spacing.

e. *Communication* Devices. The description will imply that there are eight independent devices here. If any pair of the functions is lost, a hazard arises. It is possible to combine the functions in various ways. This will reduce the number of separate pieces of equipment. However, each function must then be carried out by more than one device. It is simpler, in specification, to speak of independent devices for independent functions. The devices are:

- Forward transmitter and receiver. In platoon, the devices can distinguish signals from the vehicle ahead from all others. Otherwise, their range is at least equal to sensor range. However, at a distance, there may be several vehicles with which communication is possible.

- Rearward transmitter and receiver. These have the same specification as the forward ones, except that they react to vehicles behind.

- Lateral transmitter and receiver. These communicate with vehicles in lanes on either side. Their range is such that they can reach vehicles adjacent to those in contact with those reached by the forward and rearward communicators.

- System transmitter and receiver. These communicate with the platoon-level roadside controllers.

- Message stores and flags. There are ARs for storing messages and flags indicating their presence. When a message is received, a flag is set in an AR. The message itself is stored in another AR. If the message is received from another vehicle in the same platoon, it is also passed to the appropriate transmitter. Thus a message is passed along the length of the platoon by the regulatory level controllers. There are two exceptions to this. A message clearly cannot be passed on by the first and last vehicles of a platoon. Also, there is a message type called “acknowledge-control” (see Section 8.2.2) which is not passed on. A message containing control data is also passed to the longitudinal controller.

f. *Self-monitor.* This will be called MON. MON checks the behaviour of the other vehicle-borne controllers and devices. If any fail, an appropriate fault marker is set in the VSV. The transmitters and receivers are “looping” at all times. Thus faults here are detected at once without special action at platoon level. As a vehicle passes the places where change of lane can occur, it receives a message which, among other things gives the current loc. This enables a check to be made on the odometer. The forward sensor is checked at these points also. Here a special platoon-level-controller routine is provided.

4.6 Architecture, Level 0: Physical

The physical level, as the name indicates, is the one at which physical controls operate. An example is the way in which movements of the steering axle affect the vehicle heading. Another is the way reflections from other vehicles are interpreted by the forward sensor. In this paper these operations will be taken for granted.

5. PHYSICAL LAYOUT

Along the center of every AL there is a ***lateral reference.*** This is read by the vehicles' lateral control systems and serves to keep vehicles on track.

As explained in Hitchcock, 1991, the hazards constrain the physical layout. In an automated lane, flows of 6000 vehicles/hour or more are envisaged. Maximum flow on a manual lane is around 2000 vehicles/hour. At such high densities, we must clearly admit the possibility that one vehicle will lose speed and be rear-ended. Operation in platoons is chosen for the design because, then, if there is a such a rear-end collision it occurs at low relative speed. We call this a follower's ***collision.*** Follower's collisions were first discussed by Shladover (1979). In platoon, a follower's collision may be followed by a ***platoon crush,*** in which several vehicles successively rear-end one another at low relative speed. The resulting mass of vehicles will contain uninjured people, but will be moving at 50 mph or so. All will be well, provided there are no further collisions with platoons or the infrastructure. The following platoons in the same lane are separated by platoon spacing and so can brake to rest without collision. If, however, the platoon crush strays into an adjacent lane, it is very likely to hit some fixed bit of infrastructure, provided it is not hit by another platoon first. To prevent these high-relative-speed collisions, we must design the system with barriers, called fences, between the ALs. There are also fences between the leftmost AL and any other lanes that may be on the freeway. In order to permit change of lane there are gaps in the fences. These are called ***gates.***

It violates the safety criterion to permit a manually-controlled vehicle on the ALs. The driver cannot be compelled to keep platoon spacing from the vehicle ahead. Therefore vehicles must be admitted to the system under automatic control. They enter through a gate from the transition lane (*TL*). The TL has both manually and automatically controlled vehicles on it. However the TL exists only in the neighbourhood of the gates which permit entry and exit. We shall see later that automatically controlled vehicles on the TL must assume manual control or be brought to rest in a dormitory off the TL.

At the entry and exit gates there are system transmitters and receivers which, stimulated by a message from the driver, switch manual control off and on. At the exit gate this message is duplicated through the lateral and system communicators in case of failure of one channel on the vehicle. At every gate, including the entrances, there are also communicators at both link and platoon levels. The link level communicators receive destination messages and update the routing.

The platoon level communicators at gates "overhear" messages between vehicles concerning lane changing. On the lane to which the transfer is made, there is, by the gate, a vehicle presence detector (*VPD*). On lane from which transfer is made there is an active ***turning point,*** which is controlled by the platoon-level infrastructure controller. The turning point, when active, gives some form of electromagnetic signal which defines a precise location. If the design admits two-way gates there are VPDs and turning points on both sides. The turning point is probably a succession of electromagnets of particular polarity, but the technology used for the turning points is not important here. If a vehicle is arranging to pass through a gate, the controller inspects the VPD. If the VPD reveals that passage will not create a sideswipe, the turning point is activated. The vehicle control system detects the turning point, and commences passage through the gate. If the turning point is not activated it will remain in its original lane.

6. SYSTEM MODES

In fault conditions, it may be necessary to close a lane. It may contain a stalled vehicle, or even a crash. Less serious faults may require less fully degraded conditions. Operation can then continue, in a less effective way, which still permits the freeway to be used. The whole length of a lane between two off-gates (a ***section)*** must be in the same mode. There are some rules determining the modes admissible in one section when a nearby section is in some degraded modes. Apart from this different sections can be in different modes. We distinguish the following system modes.

a. ***Normal*** (N). This is used at all times when no faults are present.

b. ***No-Entry (NE).*** Here speeds are the same as in normal mode, but the gate controllers will not activate turning points to permit entry. NE mode is used when the section contains a vehicle which cannot communicate laterally.

c. ***Slow-Aheud (SA).*** Here speeds are reduced, but there are no other restrictions. This is used when an adjacent lane is Closed-Ahead (see below). In this case, many vehicles are trying to enter this lane. It is also used in some other fault conditions; for example, if a vehicle which cannot maintain full speed is trying to enter the lane.

d. ***Closed-Ahead (CA).*** This applies when there are sections of the lane ahead which are closed. It also applies in some other fault conditions. Speed in a CA section is reduced. Adjacent lanes are in SA mode. Gate controllers deny entry to the lane. At each gate, link-level messages advise vehicles how far they can travel in this lane and indicate a route change. If there is only one exit remaining before the stopped section is reached, speed is further reduced. If need be, vehicles stop and line up at the last gate.

e. ***Crushstop (C).*** In an emergency the system will send a message placing a section or sections into this mode and identifying a loc. There is some stationary object at this loc which blocks the lane. (For example, debris from an accident may have passed a gate.) Except in one case, all vehicles in the section immediately brake at full platoon braking and remain at rest. The exception is that vehicles within platoon spacing of the identified loc apply maximum braking. This will normally lead to in-platoon collisions. These are accepted as a lesser evil. Speed is thus reduced to the greatest extent possible. Perhaps collision with the obstacle is avoided altogether. At least the impact is ameliorated. Entry is barred. Further action is determined by the system controllers. They are likely to start by entering Stop mode (see below).

f. ***Stop (S).*** This may be initiated in the same way as C mode, except that there is no maximum braking. Alternatively, it may be entered after C mode. In this mode the system controllers can order some otherwise forbidden manoeuvres. These manoeuvres will include instructing vehicles to back up on the AL, and to exit through on-gates. The objectives will be to get emergency vehicles to an accident or breakdown, to get unaffected vehicles on their way, and generally to clear up the consequences of some untoward event. Details of all this remain to be designed.

g. ***Resume (R).*** This is used to restart a section in S, SA or CA. Again, details remain to be designed. The resumption is seen as ordered by the system controllers advised by human observation.

Modes can be degraded by the system. Reversion to normal operation is seen as requiring direct orders by a human. An exception is made for NE mode. When the faulty vehicle leaves the section, the section reverts to N mode.

7. FAULTY VEHICLES

The fault element in the VSV is set as follows:

0. This indicates no faults, and is the only value consistent with setting of the validation marker in the AL license.

1. This indicates an unreliable forward sensor. The vehicle cannot tell how far it is from the vehicle ahead of it, whether that vehicle is in the same platoon or another.

2. This indicates loss of the ability to transmit messages or data to a vehicle ahead.

3. This indicates loss of the ability to receive messages or data from a vehicle ahead.

4. This indicates loss of the ability to transmit messages or data to a vehicle behind.

5. This indicates loss of the ability to receive messages from vehicles behind.

6. This indicates loss of ability to transmit to vehicles in adjacent lanes.

7. This indicates loss of ability to receive from vehicles in adjacent lanes.

8. This indicates loss of ability to transmit to the system.

9. This indicates loss of ability to receive from the system.

10. This indicates loss of ability to locate accurately.

11. Any other fault (e.g. inability to maintain speed, inability to remain on track).

If a vehicle develops two or more faults, a hazard can arise at once. The vehicle is immediately brought to rest, and Stop mode is called. The design is such, however, that if one fault only is present, the vehicle can proceed without hazard, provided no other fault interacts. The other fault may be on anothervehicle. Therefore, the controller is such that if a fault flag is set the vehicle will immediately start a series of manoeuvres to quit the **ALs.** It may be possible for it to be driven normally on ordinary roads, so the driver need not be stranded. Further, a message, identifying the faulty vehicle is sent to the system, which keeps a list of faulty vehicles. They might decide to continue, to enter a degraded mode or to call for Stop mode. On leaving, the faulty vehicle advises the system that it has left. If the vehicle does not exit in a reasonable time the system controllers are advised, for in this position, human intervention is called for. The same is true if the system detects two faulty vehicles which are in danger of interaction.

8. OPERATION OF THE SYSTEM

We shall now describe how the system behaves as a whole, and then go on to describe how the platoon-level controllers achieve this. In the appendixes, detailed specifications of the modules making up the controllers are presented. System operations are divided into ***manoeuvres*** and probes. There are some minor other activities associated with entry and exit. The probes are concerned to discover fault conditions by using equipment which is not otherwise engaged in such a way that non-functioning can be detected. We will discuss the manoeuvres first.

There are five basic manoeuvres. Only three, however, apply in normal operation. These are, in inverse order of priority:

a. ***Merge.*** A platoon of less than optimal size joins on to another.

b. ***Split.*** A platoon divides into two, separated by platoon spacing.

c. ***Change-lane.*** A free agent (only) transfers through a gate into an adjacent AL. Entry and exit are basically change-lane manoeuvres. The process is slightly different. Control has to pass from manual to automatic (or the reverse) at the right time.

The other two manoeuvres are initiated when there is a faulty vehicle.

d. ***Emergency-Change.*** A faulty free agent moves from one lane to another on its way out.

e. ***Forced-Split.*** A platoon divides in such a way that a faulty vehicle is either at one end of the platoon or becomes a free agent. One of the probes provides means by which a non-faulty vehicle in a platoon can tell that one of its neighbours has faulty communication equipment, and therefore cannot advise it directly.

The three normal-operation manoeuvres are sufficient to enable a vehicle to do what is needed, provided it does not become faulty. It first enters the system, declares a destination, and receives a route. The route is updated every time it passes a gate. On its way through the system it will join with other vehicles to form a platoon when and if this is appropriate. When the vehicle's route requires it to change lanes or exit, split manoeuvres enable it to become a free agent. Change-lane makes the lane change or exit. Hsu and her colleagues (Hsu, et al., 1991) showed rigorously that these three manoeuvres are sufficient to meet the requirement specification if no faults arise. A similar proof that the addition of the two extra manoeuvre protocols similarly conform to an appropriately revised requirement specification would require further research.

We shall now describe the manoeuvres, the probes, and the way the vehicle-borne controller works. However, full detail will not be given here. The usual procedure will be described. An indication will also be given of what happens if the usual procedure is inappropriate. The appendixes contain formal specifications of each module. First, we shall explain the purpose of the "busy" flag. Platoon manoeuvres are controlled, in the main, by platoon leaders. Usually, a leader will engage in only one manoeuvre at once. When the manoeuvre starts, the "busy" flag is set, and a message "request_[manoeuvre]" is sent, initiating it. At the end of the manoeuvre, the "busy" flag is reset. If the message initiating a manoeuvre is received by a second vehicle, the latter's controller will consult the "busy" flag. If it is already set, the message "nack_request_[manoeuvre]" is sent. The "nack_request_[manouevre]" message acknowledges receipt of the message, but indicates inability to join in. This usually means that the would-be manoeuvrer must await completion of the existing activity. Then the request can be sent again. There is an exception to this. If a forced-split manoeuvre is requested, the existing manoeuvre, if there is one, will be suspended. This is because the forced-split request indicates the presence of a faulty vehicle in the platoon. For this reason, it may not be possible for the first manoeuvre to be completed normally.

8.1 Manoeuvres

8.1.1 Merge

Figure 2 shows a flow diagram for the merge manoeuvre. The merge manoeuvre is initiated by the leader of a platoon which is following another at a little over platoon spacing. Merge will be initiated when the following conditions are met:

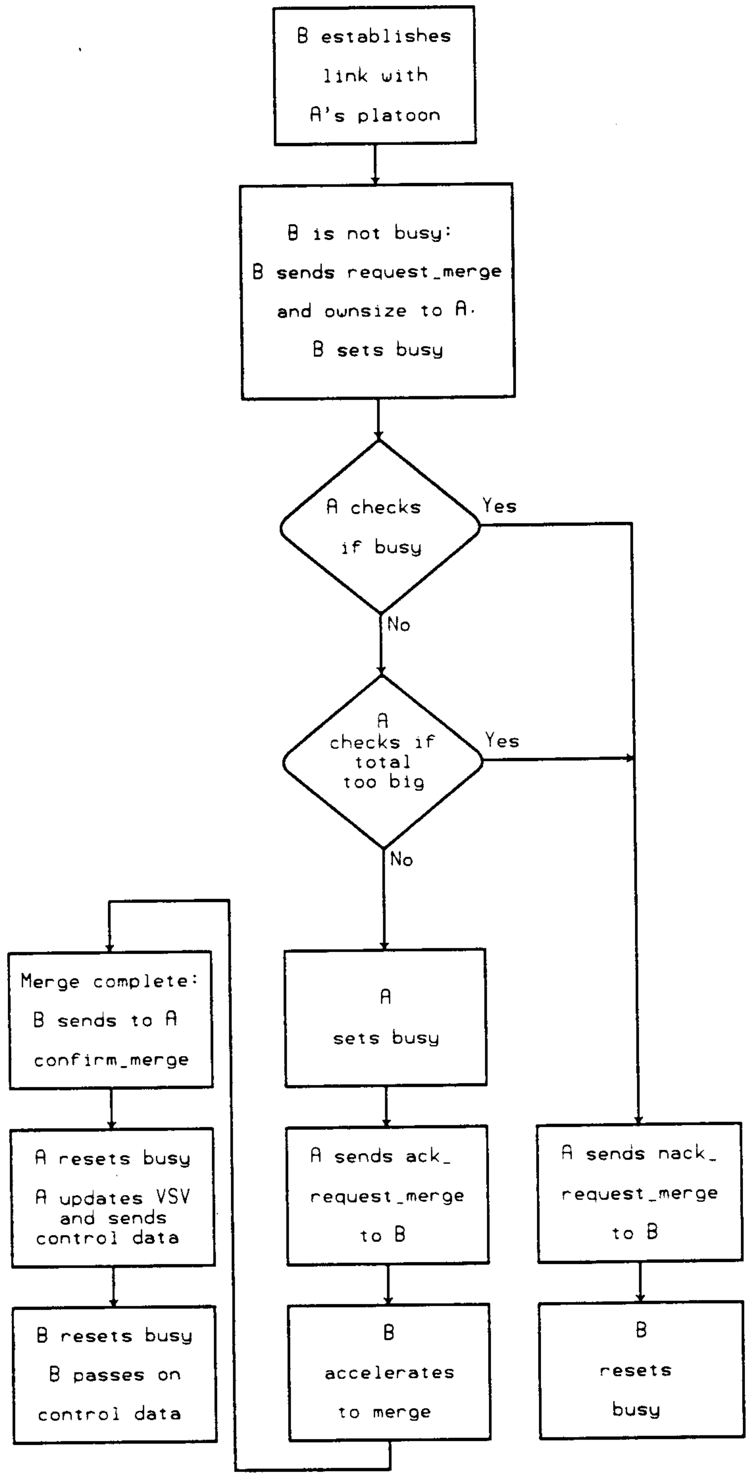

B establishes link with A's platoon
B is not busy: B sends request_merge and ownsize to A. B sets busy
A checks if busy
Yes
No
A checks if total too big
Yes
No
Merge complete: B sends to A confirm_merge
A resets busy A updates VSV and sends control data
B resets busy B passes on control data
A sets busy
A sends ack_request_merge to B
B accelerates to merge
A sends nack_request_merge to B
B resets busy

Figure 2. Merge (after Hsu, et al., 1991).

a. The leader is not otherwise busy. Its busy flag is not set.

b. The route does not call for any manoeuvre.

c. There is no fault flag newly set, nor is the “busy” flag set.

d. There is a platoon ahead in the same lane.

e. The platoon is smaller than link control’s minimum.

The sequence of events needs little description. In Figure 2, A is the leader of the leading platoon. B is the initiator, the leader of the following platoon. If A is busy, or a merged platoon would exceed the optimum size, A sends the message "nack_request_merge." Otherwise, A sends "ack_request_merge." On receipt of this B increases speed until it is a metre or so behind C, the final vehicle of A’s platoon. It then sends “confirm-merge” which is passed up the platoon to A. A responds by updating its state vector. New control data are passed the length of the new platoon. Merge is complete.

Figure 2 does not show what happens if no reply is received to "ack_request_merge." In fact a message which indicates a potential fault somewhere is sent to the system. If many such messages are received, action will be taken to resolve the problem. Figure 2 does not indicate, either, that C also sets its “busy” flag during the manoeuvre. This has no effect unless a forced-split is called in A’s platoon during the manoeuvre. Full details are in the appendixes.

8.1.2 Split

Figure 3 shows the flow diagram for the split manoeuvre. Split is usually initiated by a member of a platoon that, because its route demands it, wishes to become a free agent and change lanes. It can also be initiated by the platoon leader if the size of the platoon exceeds the optimum. This will not happen as a result of a normal merge, but it can happen if a vehicle joins a platoon following a lane change.

Part of the change-lane protocol is a manoeuvre called split-change-lane. This makes a gap in an existing platoon. When the gap reaches a gate a vehicle can change lane into the gap. Except for the manner of its initiation, split-change-lane is identical with the split manoeuvre.

Three vehicles become busy during a split. In Figure 3, A is the original platoon leader. B is the vehicle which will become leader of the new platoon. C, not shown in Figure 3, is the vehicle immediately ahead of B. If A is immediately ahead of B., there is no C. There are two

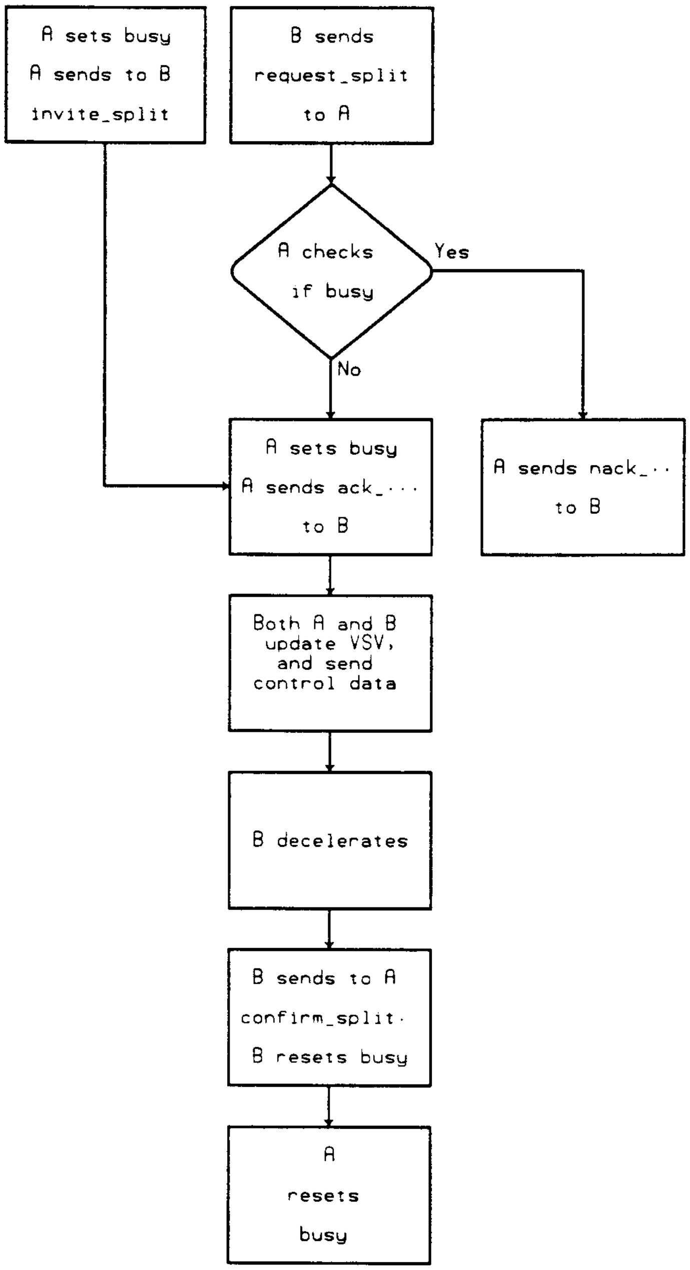


Figure 3. Split (after Hsu, et al., 1991).

variants. If A wishes to become a free agent, it sends “invite-split” to B, which is then the second vehicle in the platoon. On receipt of “invite-split” or "ack_request_split" B sets new control data, which are passed to vehicles behind it. B, and the following vehicles, decelerate until B is platoon spacing behind C. It then sends “confirm-split.” Vehicles reset “busy.” The split manoeuvre is complete.

Not shown in Figure 3 are the messages sent to the system if no reply is received to these messages. Again, the “busy” flag for C is only effective if a forced-split occurs between A and C while the split is in progress. The appendix gives details.

8.1.3 Change-Lane

Figure 4 is the flow-diagram for change-lane. Here, again, three vehicles may be involved. A is the free agent wishing to change lanes. B is, if it exists, a platoon leader or free agent in the lane into which the change is to be made. C is a platoon leader or free agent in the lane beyond B. C becomes involved only if B does not exist. A starts the procedure with the message “request-change-lane” giving its own loc, speed, and the direction of the intended lane-change. Nearby platoon leaders in the two adjacent lanes on the appropriate side of A’s lane respond with an "ack_request_change_lane." If they are busy they respond with a "nack_request_change_lane." These messages include the locs of the senders. A now works out with whom it has to cooperate. A message "thanx_but_no" is sent to all those who sent "ack_request_change_lane," but need not become part of the manoeuvre. These may be vehicles in the lane two over (i.e., potential C’s) when there is a valid response from the closer lane (i.e., a potential B). Alternatively they may be too far away from A.

If there is a B, B itself will work out the best strategy. There are three variants. First, B may request A to decelerate (message “request-decelerate”). This message refers to a gate and a time when its final member has just passed the gate. A decelerates appropriately, and as it reaches the gate it sends "confirm_dropt." The gate also receives this message and, provided the receiving side is unoccupied, A will be able to change lane. A does so, and sends “confirm-change-lane." This initiates a merge procedure, as already discussed. When the merge is complete, change-lane is also complete.

Second, B may send “request-split-change-lane” to one of the members of its own platoon. The split takes place as already discussed. The message “confirm-split” will be received by B in due course. As B reaches the gate, it sends "confirm_split_change_lane" to A. This message includes a gate and a time at which the change will be made. Meanwhile, A has been maintaining speed, and should therefore reach the specified gate at or very near the stated time. The gate reacts to the message also. A changes lane. A’s message “confirm-change lane” starts a merge procedure, as already discussed.

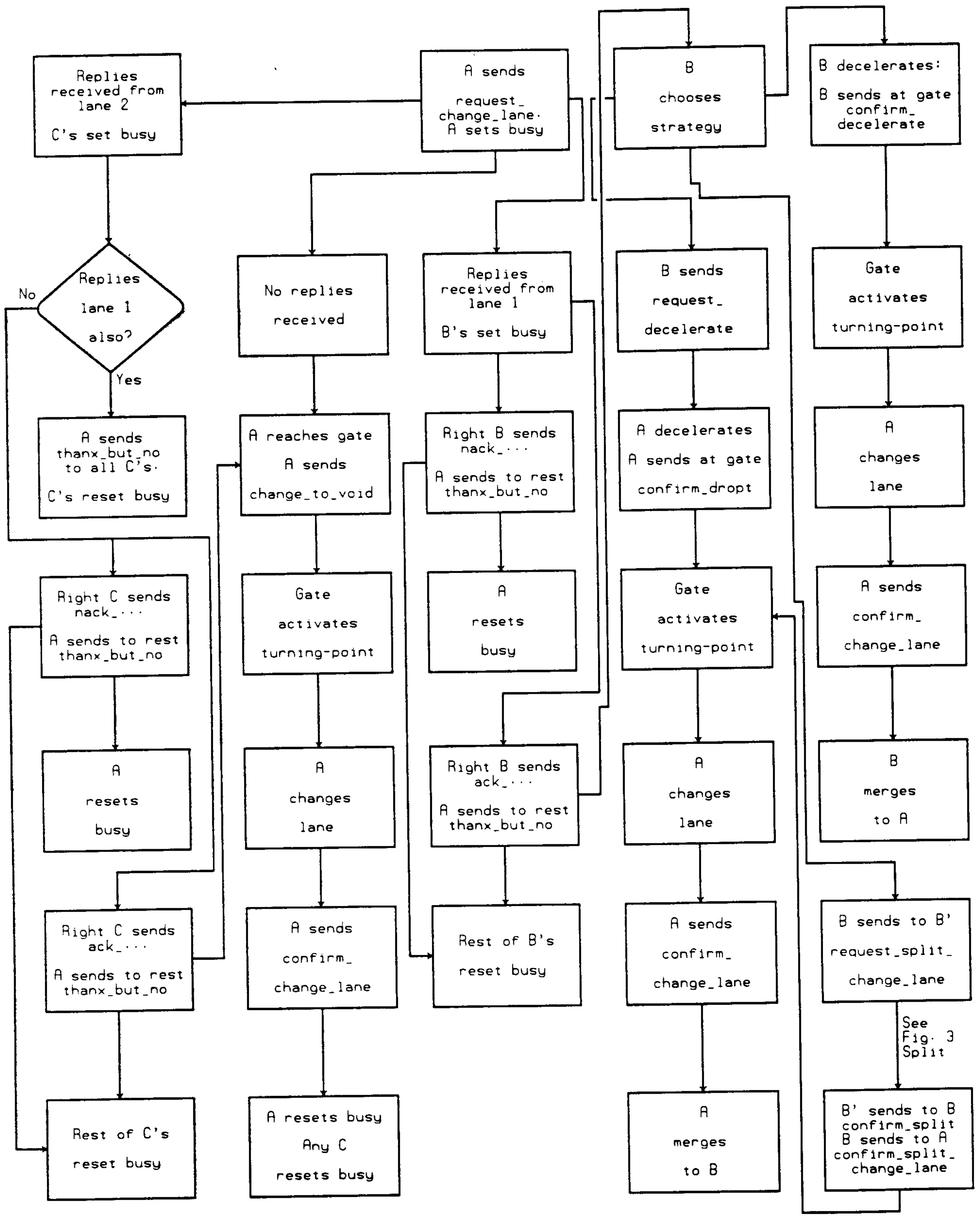

Replies received from lane 2
C's set busy
A sends request_change_lane. A sets busy
B chooses strategy
B decelerates:
B sends at gate confirm_decelerate
Replies lane 1 also?
No
Yes
No replies received
Replies received from lane 1
B's set busy
B sends request_decelerate
Gate activates turning-point
A sends thanx_but_no to all C's.
C's reset busy
A reaches gate
A sends change_to_void
Right B sends nack_...
A sends to rest thanx_but_no
A decelerates
A sends at gate confirm_dropt
A changes lane
Right C sends nack_...
A sends to rest thanx_but_no
Gate activates turning-point
A resets busy
Gate activates turning-point
A sends confirm_change_lane
A resets busy
A changes lane
Right B sends ack_...
A sends to rest thanx_but_no
A changes lane
B merges to A
Right C sends ack_...
A sends to rest thanx_but_no
A sends confirm_change_lane
Rest of B's reset busy
A sends confirm_change_lane
B sends to B' request_split_change_lane
See Fig. 3 Split
Rest of C's reset busy
A resets busy
Any C resets busy
A merges to B
B' sends to B confirm_split
B sends to A confirm_split_change_lane

Figure 4. Change-Lane
(Change is to lane 1: platoon-leader B is in lane 1.)
(lane 2 is adjacent to lane 1: leader C is in lane 2.)

Finally B may itself decelerate. Having done so, B sends “confirm-decelerate” as it reaches the gate. Again, this message names a gate and a time, which will be just ahead of B. Again A’s message “confirm-change-lane” starts a merge procedure as already discussed. Change-lane is complete.

If there is a C, all that is required of it is that it hold its relative position to A. In particular it will not change lane itself. Furthermore, it occupies the position where a vehicle in its lane would have to be for a change-lane to interfere with the changer. A will send the message “change-to-void” which names the gate through which A will change lane, as well as the time of the change. The specified gate will activate the turning point, provided there is no obstacle to doing so. A will pass through the gate, and will send “confirm-change-lane.” Both A and C will reset “busy." Change-lane is complete.

If there is neither a B nor a C, A must still send “change-to-void” to activate the gate. A changes lane and the manoeuvre is complete.

Not shown in Figure 4 are the actions taken if no reply is received after some time. Under these conditions a message is sent to the system, and the manoeuvre is canceled. It may be restarted later. If, when it has changed lane, A finds itself too far away from the platoon with which it is trying to merge, a message is again sent to the system. The most likely reason for this is that one vehicle has developed a fault in the odometer. Finally, if during the earlier stages of the manoeuvre forced-split is called, change-lane is called off. If however, the change of lane is already committed, the following merge continues. All these extra details are discussed in the appendix.

8.1.4 Emergency-Change

Figure 5 is the flow diagram for emergency change. Emergency-change is initiated by a faulty vehicle which has become a free agent. Such a vehicle is now programmed to leave the ALs as soon as possible. Because the vehicle is faulty, the usual change-lane procedure is inappropriate. Change-lane requires several subsystems, any of which may be the site of the fault. Change-lane involves close approach to other vehicles and formation into new platoons, which is also inappropriate when a vehicle is faulty.

Emergency-change involves cooperation with four other platoon leaders. These are requested to run a platoon spacing ahead, and a platoon spacing behind the faulty vehicle. Two of the four are in the lane into which the change will be made. The other two are in the lane beyond. All simply maintain themselves in this position while the change is made into a large gap. The four shepherding vehicles adopt the speed, whatever it may be, of the faulty vehicle. When the

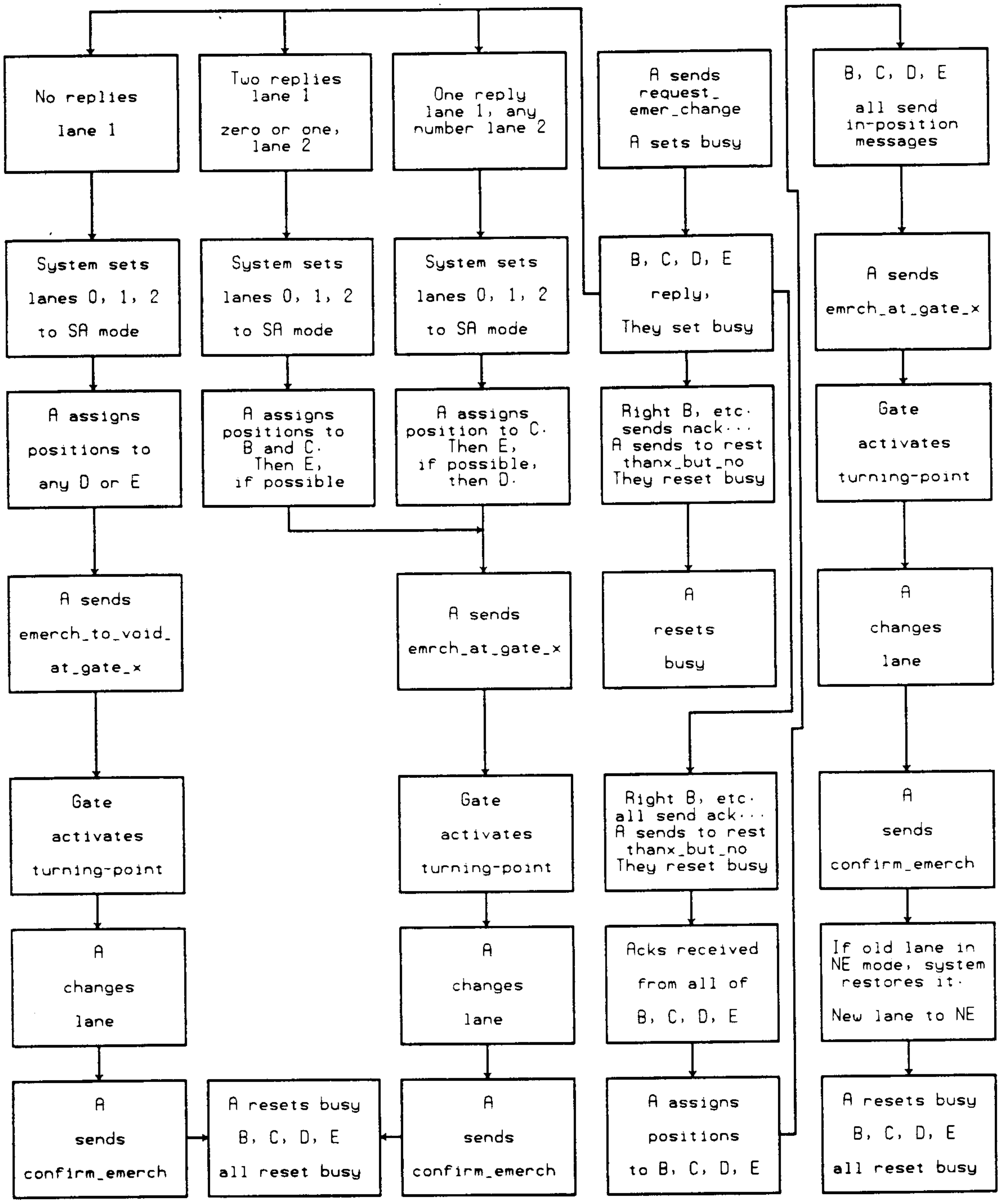

No replies lane 1
Two replies lane 1 zero or one, lane 2
One reply lane 1, any number lane 2
A sends request_emer_change A sets busy
B, C, D, E all send in-position messages
System sets lanes 0, 1, 2 to SA mode
System sets lanes 0, 1, 2 to SA mode
System sets lanes 0, 1, 2 to SA mode
B, C, D, E reply, They set busy
A sends emrch_at_gate_x
A assigns positions to any D or E
A assigns positions to B and C. Then E, if possible
A assigns position to C. Then E, if possible, then D.
Right B, etc. sends nack... A sends to rest thanx_but_no They reset busy
Gate activates turning-point
A sends emerch_to_void_at_gate_x
A sends emrch_at_gate_x
A resets busy
A changes lane
Gate activates turning-point
Gate activates turning-point
Right B, etc. all send ack... A sends to rest thanx_but_no They reset busy
A sends confirm_emerch
A changes lane
A changes lane
Acks received from all of B, C, D, E
If old lane in NE mode, system restores it. New lane to NE
A sends confirm_emerch
A resets busy B, C, D, E all reset busy
A sends confirm_emerch
A assigns positions to B, C, D, E
A resets busy B, C, D, E all reset busy

Figure 5. Emergency Change-Lane.
(Change is to lane 1: B and C in lane 1, B leading)
(lane 2 is adjacent to lane 1: D E in lane 2, D leading)
SA mode = Slow-Ahead, NE = No-Entry

vehicle has changed lane it sends "confirm-emergency-change. " Busy flags are reset. The shepherds are free to engage in other manoeuvres.

There are a number of special features, activated in particular circumstances. The fault may be in a lateral communicator (fault 6 or 7). If so, the lane in which it is running is in NE mode. When the change is made, the clear lane reverts to N mode. The one which now contains the faulty vehicle is put to NE. Another feature which comes into play when fault 6 or 7 is present is that the system communicators come into play. Messages which cannot be transmitted or received by the lateral communicators are relayed through the system communicators. As will be seen, the gate controllers communicate with both the lateral and system communicators, so that if a vehicle is faulty in either the lateral or the system communicators the non-faulty one of the pair passes the message to activate the turning point.

If forced-split is called in either shepherding platoon, the manoeuvre is broken off. We now have two faults in different vehicles interacting. The system controllers are advised. This is an area where hazards can be readily generated and human supervision is therefore necessary. Full details are given in the appendixes.

8.1.5 Forced-Split

The forced-split manoeuvre is initiated by a vehicle in a platoon which develops a fault. A faulty vehicle has to quit the ALs. The first step is to become a free agent. Usually, the reaction is to call for a split ahead so that the faulty vehicle becomes platoon leader. If, however, the fault is in the rearward communicators, the initial split is behind the faulty vehicle. If the faulty vehicle is platoon leader, the split is also behind it. We shall see later that even if the fault prevents full communication with other vehicles in the platoon, the probes enable the other vehicles to operate the forced-split.

Forced-split differs from split in these ways:

a. A confirm-forced-split message is sent when the manoeuvre is complete. However, no reliance is placed on its being received. After a reasonable interval of time, the manoeuvre is terminated.

b. The original platoon may send a message breaking off some other activity; if not, this leader checks, when the forced-split is over, to see if another activity should be resumed.

c. The last vehicle in the rear platoon will receive data indicating that it is a member of a new platoon. It consults it own "busy" flag. If the flag is set, another manoeuvre is in progress. A message is sent to the new platoon leader indicating as much.

Because these differences do not show up well in a flow diagram, none is shown for this case. Figure 3 applies here too.

Forced-split differs from other manoeuvres in that it overrides the "busy" flag. A faulty vehicle in a platoon is either an imminent danger causing a crash (probably a low speed one) or interferes with the execution of the manoeuvre. Therefore the manoeuvre is preempted.

In the case of a split or merge manoeuvre, the manoeuvre continues in the unaffected platoon. The leaders in a platoon which is engaged in a forced-split manoeuvre can receive messages. If they do, however, the message is stored and is not responded to until the forced-split is complete. Then the system recovers stored messages and either sends the "confirm_[manoeuvre]" message or continues with a merge or split. If the forced-split occurred in the first of the two platoons engaged in a merge or split, the original leader of this platoon is now inaccessible. The last member, however, does have "busy" set, and has stored the necessary data about the previous manoeuvre. This last member sends this forward to the new leader. An exception arises when the faulty vehicle is itself the last member of the platoon. Now a merge manoeuvre will be terminated. A message is passed to the following platoon to fall back at the time the forced-split is initiated.

If forced-split is called in a cooperating platoon, a change-lane manoeuvre is terminated unless the change of lane has already taken place, or the gate has been activated for the change. At this stage in a change-lane manoeuvre, a merge is in progress or about to begin. It will continue as other merges do.

Further detail about this manoeuvre is given in the appendix. Formal specifications of all modules are included.

8.2 Probes

8.2.1 Platoon Leader Probe

Figure 6 gives a flow diagram for the platoon leader probe. The platoon leader probe is a check on the continued function of the forward sensor. The probe is initiated whenever a platoon leader reaches a gate, unless it is "busy." If the forward sensor perceives at least one vehicle ahead, a message is sent giving the loc of the sender and indicating the distance of the vehicle perceived. A vehicle receiving this message may reply if it is the last vehicle in a platoon. There may be many such vehicles. Each will reply if its own loc makes it likely that it is the

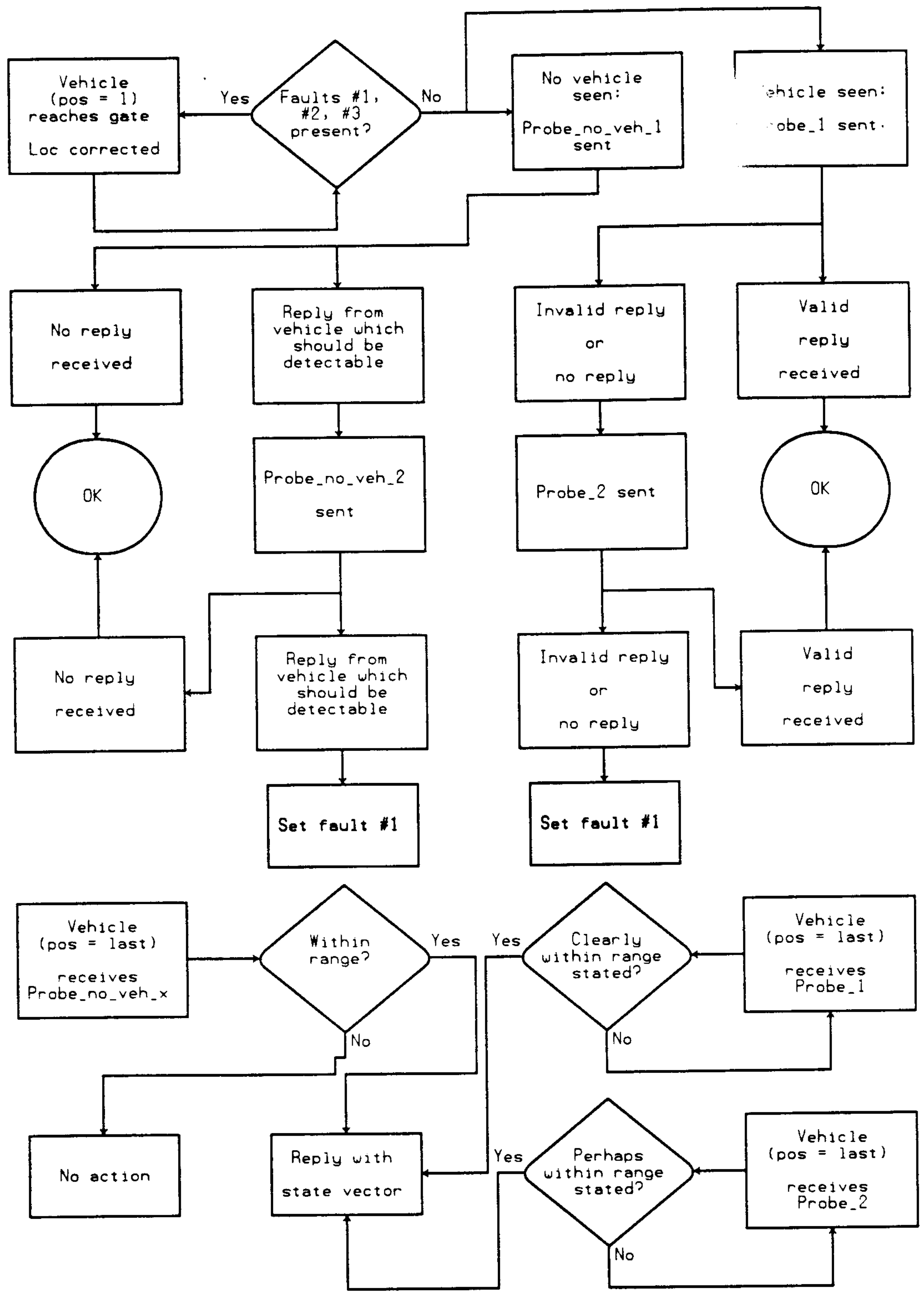

Vehicle (pos = 1) reaches gate Loc corrected
Yes
Faults #1, #2, #3 present?
No
No vehicle seen: Probe_no_veh_1 sent
ehicle seen: obe_1 sent
No reply received
Reply from vehicle which should be detectable
Invalid reply or no reply
Valid reply received
OK
Probe_no_veh_2 sent
Probe_2 sent
OK
No reply received
Reply from vehicle which should be detectable
Invalid reply or no reply
Valid reply received
Set fault #1
Set fault #1
Vehicle (pos = last) receives Probe_no_veh_x
Within range?
Yes
No
Yes
Clearly within range stated?
No
Vehicle (pos = last) receives Probe_1
No action
Reply with state vector
Yes
Perhaps within range stated?
No
Vehicle (pos = last) receives Probe_2

Figure 6. Platoon Leader Probe.

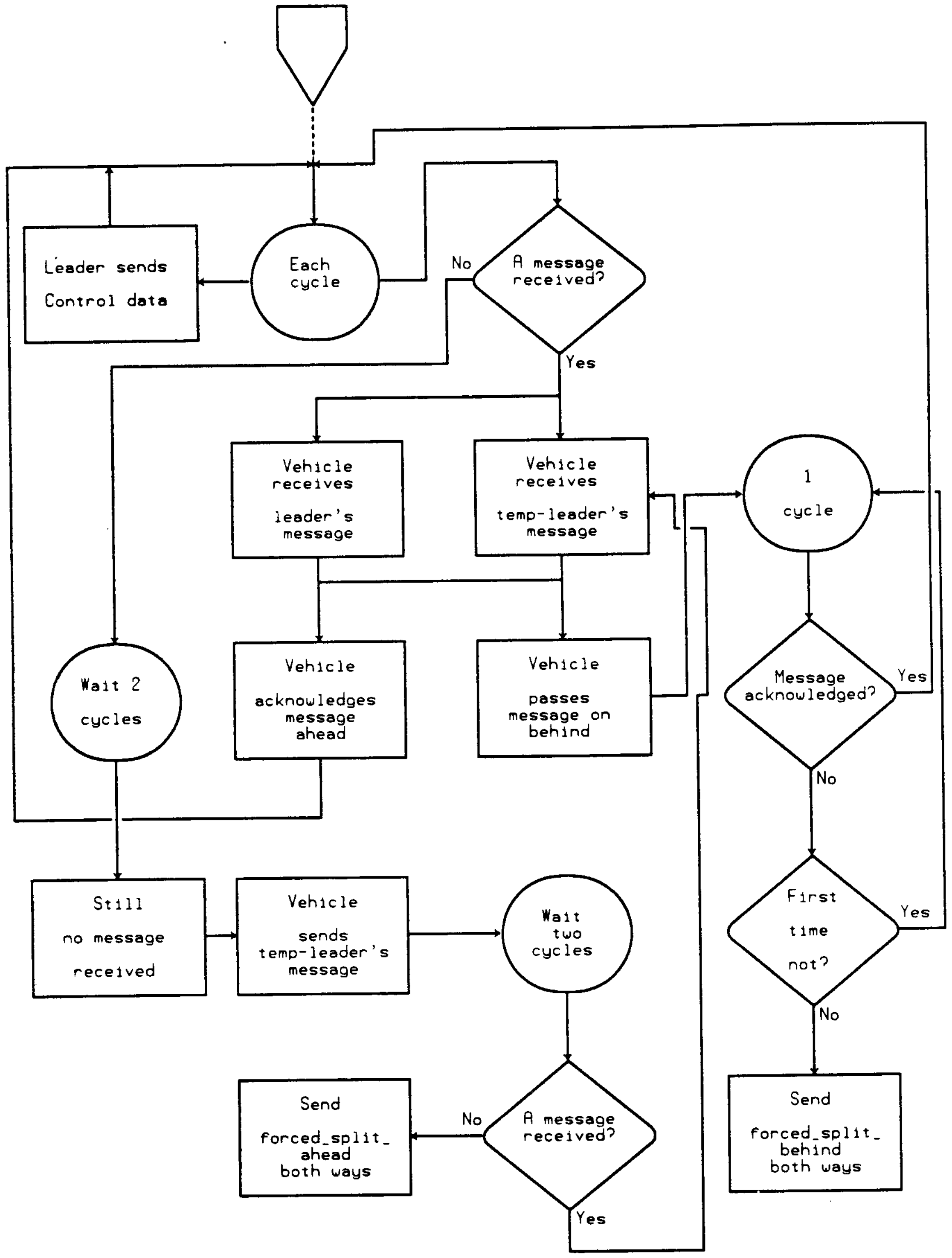
vehicle addressed. If the sender receives a reply from a vehicle whose loc makes it likely that it is the one seen, all is well.

If, however, no reply is received by the first sender, another message is broadcast. This message calls for a reply from any vehicle credibly being addressed. This is because, if the road is strongly curved, it may be difficult to be precise about the relative position of locs on diverse lanes. If, again, a reply is received and appears to be valid, all is well. Otherwise there are two possibilities. Either the sender is seeing things which are not there, or the receiver, is unable to communicate. If there is or seems to be more than one potential receiver, the first alternative is valid. Fault #1 is set by the probing vehicle and the system is advised. If there is or seems to be just one potential receiver, a message is sent to the system. The system may reply that indeed, there is a vehicle in this general area with such a fault. If so, no further action is taken. Otherwise fault #1 is set by the prober. The other possibility is that no vehicle is perceived, in which case a message is broadcast to any potential receiver ahead saying so. If a reply is received, one tries again. If again a reply is received, fault #1 is set. The system is informed.

8.2.2 In-Platoon Probe

Figure 7 is a flow diagram of the in-platoon probe. If a vehicle in a platoon develops a communication fault, it will set a fault flag. From its side, the faulty vehicle will start a forced-split. However, if the fault is in the forward or rearward communication system, the vehicle with which communication has failed may not be aware of the need to forced-split. The in-platoon probe deals with this. Every time a message with control data is sent with the leader's VSV data down the platoon, each vehicle responds ahead with an "acknowledge-control" message. As explained in Section 4.5, this message, uniquely, is not passed on up the platoon. It is recognized that a number of temporary conditions will interfere with the passage of one message. Two consecutive failures to receive a control message are required before the vehicle deduces that there is a failure ahead of it. It then sends a temporary leader's message, giving its own control data. The would-be leader may subsequently receive a message from another temporary leader ahead of it. This can occur when the break in communication is not immediately ahead of it. If a temporary leader receives a message from ahead, it adopts a follower's role again. If not, it will send a message in both directions, indicating that a forced-split will be formed ahead of it.

Alternatively, there may be no acknowledgment. Once again, confirmation that the loss is permanent is awaited. After that, a message is sent, both ways, indicating that a forced-split will occur behind the vehicle sending the message.



Leader sends Control data
Each cycle
No
A message received?
Yes
Vehicle receives leader's message
Vehicle receives temp-leader's message
1 cycle
Wait 2 cycles
Vehicle acknowledges message ahead
Vehicle passes message on behind
Message acknowledged?
Yes
No
Still no message received
Vehicle sends temp-leader's message
Wait two cycles
First time not?
Yes
No
Send forced_split_ ahead both ways
No
A message received?
Yes
Send forced_split_ behind both ways

Figure 7. In-Platoon Probe.
(The probe operates cyclically: strictly, there is no entry point. Comprehension is eased by starting at the point indicated.)

8.3 Entry and Exit

Entry has two components. First the vehicle must signal the system that it wishes to join the system, and give a destination. The system will reject the vehicle's request for entry if it does not signal that its AL license is valid. It will also reject the request if the destination is inaccessible. This could happen if the driver is trying the wrong direction, or if there is some breakdown ahead.

If a vehicle is accepted, the next stage is a change-lane. This differs in two ways from other change-lane manoeuvres. The driver, before requesting entry, can match speed and position so as to make entry simpler. Also, should entry prove to be impossible for any reason, the vehicle will ultimately pass the last on-gate. It will now be treated as if it had just exited. The driver will be invited to resume manual control.

Exit, too, starts with a change-lane, like any other, on to the TL. However, in this case, the system sends a warning to be ready to resume manual control on exit. Immediately after exit is confirmed by the VPD, the driver is offered the opportunity to resume manual control. If manual control is not taken, the vehicle will be first slowed down as it passes a particular communicator near the end of the TL. If it reaches the end of the TL, it will be brought to rest in a dormitory.

8.4 Operation of the Vehicle-Borne Controller

The controller has many functions, and cannot remain with any operation for a significant period. Instead it cycles, continually, through six functions. If any of these is active, the controller will enter it. The controller will initiate some operation, if it is time to do so. Otherwise, the controller will simply set a flag. When control returns on the next cycle, the controller will pick up the operation where it left off. In the interval it has examined the need for operation in the other five categories.

All this is set out in detail in the appendixes. Figure A. 1 shows a schematic, naming the modules. A full description of the latter will be found in the appendix "Supervisor." The six elements operate as described below. We have given the elements mnemonic names.

a. ***Buschek.*** "Buschek" is "check busy." If the busy flag is reset, control passes on. Otherwise the controller consults flags to discover which manoeuvre is in progress. Another flag indicates the role played by this vehicle, while a third records the stage reached.

b. ***Forprob.*** "Forprob" is "Forward probe." This section progresses the platoon leader's probe. Again, flags are set, indicating that the probe is in progress. Another flag advises how far the probe has progressed.

c. ***Messrec.*** "Messrec" is "Messages received" This module is responsible for initiating new actions in response to messages. Flags have been set in the message register by the regulatory level communicator controllers indicating which messages have been received. This level will also already have passed on many up or down platoon. Regulation level will also have discarded messages which do not apply to this vehicle. Other messages are of concern to particular elements, and their flags are noted there.

However, some messages are initiators. "Request_merge" is an example. Another is "Probe-l," the first forward probe message from a vehicle to the rear. "Messrec" notes these. It examines each incoming message in turn, and transfers control to appropriate elements.

d. Ne*wfalt*. "Newfalt" is "New fault." Fault flags are inspected to see if any are set. The fault-l probe is called from here. This probe is used by a vehicle which has lost forward sensor capacity, and has no vehicle ahead of it in platoon. It calls forward giving its loc. Vehicles ahead in the same lane give their own locs. These data are passed to the longitudinal control system instead of the sensor readings. Here too, messages are sent if a vehicle is for some reason moving very slowly.

e. ***Actcalled.*** "Actcalled" is "Calls for Action." If faults are present, the appropriate manoeuvre is called. If there are none, the module examines the routing to see if any manoeuvre needs to be initiated. If the platoon is above or below the optimum range, it may be appropriate to send a "request-merge" or "request_split." At a gate, it may be appropriate to initiate the platoon-leader's probe.

f. ***Contdat.*** "Contdat" is "Control data." The control data are passed along the platoon by the regulation level which also makes them accessible to longitudinal control. However, the in-platoon probe also needs these data, and the logic of this probe is carried out in Contdat.

This completes the general description of the system. For further details the appendix should be consulted.

8.5 Operation of the Roadside Controller

In fault-free operation, the only activities of the roadside controllers are to send signals causing manual and automatic control to be switched, and to activate the turning points at gates. In fault conditions, the system communicators also act as a backup to the normal ones. The gates communicate both with the lateral and with the system communication subsystems. The system is called on, whenever there may be a vehicle that cannot communicate over a distance. (The description of the platoon leader's probe in Section 8.2.1 contains an example.) Also, the system maintains two lists of faults and potential faults. Whenever a fault flag is set, the roadside controller is advised. When a faulty vehicle exits it is removed from the list. The system controllers are advised if there is too long an interval before a faulty vehicle exits. These people also receive a message if there are several faulty vehicles close together.

Further, there are a number of events which are reported to the system which indicate that something is faulty somewhere, but do not enable a culprit to be identified. Excessive delays or failures to respond in a merge process are examples. Another example is a vehicle changing lane too close to or too far away from a partner in the change or emergency-change protocol. Such events result in vehicles being put on a potential-fault list. If they appear several times, a message is sent declaring them faulty.

9. WORK OF HSU, et al. (1991)

It has already been explained (see Section 1) that this specification is the completion of one due to Hsu and her collaborators. These authors have agreed to their work being used in this way. While their work is not a large part of what is described here, it did, of course, inspire the whole. In particular Hsu and her colleagues did not specify a physical layout. They did not describe the cyclic action of the vehicle-borne controller. Two of the manoeuvres (forced-split and emergency-change) have been added. So have both the probes. The whole of the treatment of non-reply to messages described here is new.

However, the small fraction of the pages do contain a vision of how a system might work that permeates the whole of this report. This work does not address the same topics as HSU, et al. Its conclusions are, it is believed, useful and are not trivial consequences of the previous work. Nevertheless, this work has been greatly aided by being able to build on the earlier work.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the technical support and encouragement of Dr. Steven E. Shladover, Technical Director, PATH. The author experienced some difficulty in writing this paper as clearly understood prose. The comments of Mr. Sompol Chatusripitak of Caltrans were very helpful.

The great value of the work of the authors of Hsu, et al. (1991) has already been acknowledged.

REFERENCES

Hitchcock, A. 1991. "Intelligent Vehicle/Highway System Safety: Problems of Requirement Specification and Hazard Analysis," Transportation Research Board Annual Meeting, Washington, D.C.

Hitchcock, A. 1992(a). "Methods of Analysis of IVHS Safety," PATH Research Report UCB-ITS-PRR-92-14, Institute of Transportation Studies, University of California, Berkeley, CA.

Hitchcock, A. 1992(b). "Fault Tree Analysis of an Automated Freeway with Vehicle-Borne Intelligence," PATH Research Report UCB-ITS-PRR-92-15, Institute of Transportation Studies, University of California, Berkeley, CA.

Hsu, A., Eskafi, F., Sachs, S., and Varaiya, P. 1991. "The Design of Platoon Maneuver Protocols for IVHS," PATH Research Report UCB-ITS-PRR-91-6, Institute of Transportation Studies, University of California, Berkeley, CA.

Shladover, S.E. 1979. "Operation of Automated Guideway Transit Vehicles in Dynamically Reconfigured Platoons," Urban Mass Transportation Administration Report UMTA-MA-06-0085-79-1, 2 & 3, Springfield, VA.

Varaiya, P., and Shladover, S.E. 1991. "Sketch of an IVHS Systems Architecture," PATH Research Report UCB-ITS-PRR-9 l-3, Berkeley, CA.

APPENDIXES

A.1 INTRODUCTION

The appendixes contain formal statements of the form and effect of each module. The form selected is loosely based on the statements required in a number of formal computer languages. There is, however, no computer language known which can accept these statements. Since many of the terms used are not axiomatized, a great deal more would be needed before these formal statements could be so used. However, this form has been found to be adequate for manual verification of the statements made in the course of a fault tree analysis, and this is the application intended.

The general section of these appendixes contains the modules which refer to more than one vehicle or platoon. The others are grouped into sections. Each section refers to the supervisor itself (Figure A. 1) or to one of its six elements. However, the figures for the element Busycheck are broken down into the five manoeuvres (Figures A.2 - A.6). At the end, the “System” section describes the operation of the platoon-level roadside controller.

The following abbreviations are used for the system modes:

N — Normal	NE — No Entry	SA — Slow-Ahead
CA — Closed-Ahead	S — stop	C — Crashstop
R — Resume		

A.2 HOW TO USE THE MODULE DEFINITIONS

If it is wished to determine exactly how the system, or some part of it, works, it is necessary to consult the written module definitions. The figures provided here do act as a guide to the flow of control through the modules. However, they are not a full flow-chart. Where a module is at a branch, for example, the figures do show two successors. They do not say what the branching criterion is. That is done in the written module definitions. Similarly, a branching criterion may require external input. The figure may show that there is a predecessor which provides the relevant data — but the message itself is only identified in the text. Equally the words within a box in a figure are necessarily abbreviated, and often omit relevant information.

In general, as each of the elements in the main routine (called supervisor) is entered, the first action is to consult flags. The flags indicate whether an operation is in progress, and if so, the point which it has reached. Thus, for example, if Busycheck is entered, the system first checks the busy flag. If the flag is set, a manoeuvre is in progress. Alternatively, the flag may just have been set by Actcalled. In either event, the system first determines the role the vehicle is playing. Flags with names like split(2) or emerch(3) indicate the vehicle’s role. In fact a vehicle with split(2) set is the leader of the second platoon in a split manoeuvre. Emerch(3) indicates the leader of the trailing platoon in the lane which is to be entered in an emergency change. Within this role, there may be stages. Thus with split&) set, and the stage flag = 3, the newleader is dropping back to platoon spacing. The action is to consult the reading of the forward sensor, and determine if the full separation is yet achieved.

So much is perhaps reasonably clear from the diagrams. But for full understanding, the text must be used. An example may be useful. Suppose that a platoon is alone on the road except for a vehicle ahead in the same lane which has fault #5. The vehicle thus cannot receive a message from the leader of our platoon. A gate is reached. When the element Actcalled is reached, the forward probe is started. To see how consult Figure A.lO. In the circumstances described Actcalled will successively call the modules Linkmess, Mersplit and Startprobe. By looking at each specification it is apparent that Startprobe will set the stage flag in Forprobe = 1.

Now examine Figure A.7. It is perhaps unnecessary to examine the specifications of the modules Forprobe, Insight, Recackprobel, Probeagain, and Recackprobe2. On successive calls to Forprobe, as the specifications will confirm when examined, the flag advances from 1 to 8, Probe-l is sent (flag = 1) and later Probe-2 (flag = 5) is sent. Since the vehicle addressed does not receive the message there is no reply. At flag = 8, Recackprobe2a is called. No message has been received. However there is only one vehicle in sight. So the flag is set to 16, and the message Fault_1_veh is sent to the system. Now consult the system section. The system will consult its list of faulty vehicles. It will discover that there is indeed a vehicle with fault 4 or 5 in the area from which the message is sent. So it sends No-fault-l. Whatever the relative cycles of system and the vehicle controller, there is sufficient time for this message to be picked up when the flag is 16, 17 or 18. Nofault then resets the flag in Forprob. No fault is wrongly ascribed to our platoon leader.

A.3 GENERAL

These modules are general They are called from several locations, and have no specialised successors:

Name: Cacall **Fig:** 8
Admitted in: N NE SA
Input: Locations
Requires: -
If branch: Yes **Condition:** In range stated? Pos?
Effect: 1. If within range stated, maxspeed to reduced speed; update state vector; if pos = 1, full platoon braking;
2. Endif; endif; return;

Name: Callstop **Fig:** 6, Gen
Specification in: Emerch (6)

Name: Crashstopbehind **Fig:** -
Admitted in: N NE SA CA
Input: -
Requires: Several precursors
If branch: No **Condition:** -
Effect: Message to system - systems sets crashstop mode from pos of Vehicle calling, back twice interplatoon spacing; sets CA mode behind that and SA mode in parallel lanes; inform system supervisors;

Name: Crashstopcall **Fig:** 8
Admitted in: N NE SA CA
Input: Locations, and perhaps name of platoon
Requires: -
If branch: Yes **Condition:** Within platoon spacing of loc named? In range stated? Pos?
Effect: 1. If in named platoon, or within platoon spacing behind loc stated,
2. Maxspeed to zero; full braking;
3. Endif; If within range stated, maxspeed to zero; if pos = 1, full platoon braking;
4. Endif; endif; return;

N a m e : Cutspeed **Fig: 8**
Admitted in: N NE SA CA
Input: Locations, speed
Requires: -
If branch: Yes **Condition:** In range stated? Pos?
Effect: 1. If within range stated, maxspeed to stated speed; store original maxspeed;
2. Endif; return;

Name: Maystop **Fig: 9**
Admitted in: N NE SA CA
Input: Locations
Requires: -
If branch: Yes **Condition:** In range stated? Pos?
Effect: 1. If within range stated, set flag in Newfalt;
2. Endif; return;

Name: Necall **Fig: 8**
Admitted in: N NE SA
Input: Locations
Requires: -
If branch: Yes **Condition:** In range stated? Pos?
Effect: 1. If within range stated, update state vector;
2. Endif; endif; return;

Name: Norcall **Fig: 8**
Admitted in: NE SA
Input: Locations
Requires: -
If branch: Yes **Condition:** In range stated? Pos?
Effect: 1. If within range stated, maxspeed to stored value; update state vector;
2. Endif; return;

Name: Scall **Fig: 6,** Gen
Specification in: Emerch (6)

Name: Sacall2 **Fig: 8**
Admitted in: N NE
Input: Locations
Requires: -
If branch: Yes **Condition:** In range stated? Pos?
Effect: 1. If within range stated, maxspeed to reduced speed; store original maxspeed; update state vector; if pos = 1, full platoon braking;
2. Endif; endif; return;

Name: S topcall	**Fig: 8**
Admitted in: N NE SA‘CA	
Input: Locations	
Requires: -	
If branch: Yes	**Condition:** In range stated? Pos?

Effect: 1. If within range stated, maxspeed to zero; if pos = 1, full platoon braking;
2. Endif; endif; return;

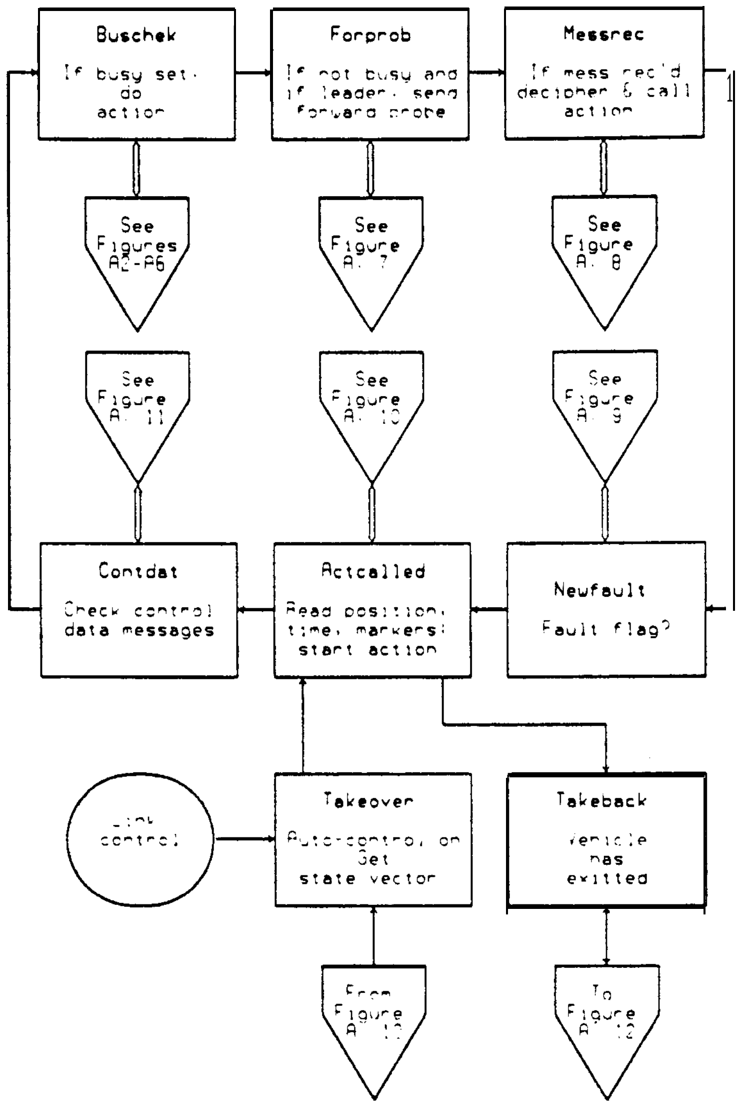


Figure A. 1. Supervisor.

A.4 SUPERVISOR (Figure A.l)

Name: Actcalled **Fig:** 1, 10
Admitted in: **N NE** SA CA
Input: Messages from link control
Requires: Newfalt or Takeover
If branch: Yes **Condition:** Fault present? Message from link? Platoon too big, small? Time?
Effect:
1. If a fault flag is set, then if emerch(1) is set, or forspl(x), x = 1 -5 is set, ;
2. Else send to system fault_#x_present with vector; if ownsize > 1, then if fault #4 or #5 is present,
3. Call Fault4or5;
4. Else if pos = 1, call Forspll;
5. Else call Forsplal ;
6. Endif; endif; else call Emerchcall;
7. Endif; endif; else call linkmess;
8. Endif; Go to contdat;

Name: Buschek **Fig:** 1,2,3,4,5,6
Admitted in: All
Input: -
Requires: Contdat (in iterating cycle)
If branch: Yes **Condition:** Busy? Internal flags?
Effect:
1. If busy set, examine flags in order shown until one found set; then, if need be, examine flag2, and call routine shown in list;
2. Endif; goto Forprob;

Flag I	*Flag2*	*Routine to call*	*Fig*
Chng(1)	1,2,3	Repwait1	4
	4	Voidch	4
	5	Recreqdecel	4
	6	Pass2	4
	7	Pass3	4
	8	Pass 1	4
	9	Sendconchll	4
	10	Sendconchl2	4
	11	Sendconchl3	4
	12	Sendcondropt	4
Chng(2)	1,2,3	Repwait2	4
	4	Compposns	4
	5	Reccondropt	4
	6	Sendcondecel	4

Name: Buschek (continued)

Flag 1	*Flag 2*	*Routine to call*	*Fig*
Chng(2)	7	Recconchl2	4
	8	Recconchl 1	4
Chng(3)		Recconchl3	4
Emerch(1)	1,2,3,4,5	Repwate	6
	6	Voidemerch	6
	7	Compposn	6
	8	Recadjahead	6
	9	Passemerch	6
	10	Neset3	6
Emerch(2,3,4,5)	1,2,3,4,5	Repwate1	6
	6	Recposn	6
	7	Inposn	6
	8	Recconemer	6
Forspl(1)		Qfaltl	5
Forspl(2)	1	Newlead	5
	2	Sepmeasfsl	5
	3	Qslowmode	5
	4	Testactflag	5
	5	Splitagain	5
Forspl(3)		Recfsplitover	5
Forspl(4)	1	Recconfspl2	5
	2	Recconfspl3	5
ForspI(5)	1	Recconfspl4	5
	2	Recconfspl5	5
Forspl(6)		Qmess	5
Merge(1)	1	Recackreqm	2
	2	Recunmerge	2
	3	Norepm3	2
Merge(2)		Recconmer	2
Merge3		(No action)	2
Split(1)		Rconspl	3
Split(2)	1	Recackreqspl	3
	2	Newlead	3
	3	Sepmeas-s	3
Split(3)		Recsplitover	3

Name: Contdat **Fig:** 1, 11
Admitted in: N NE SA CA
Input: -
Requires: Actcalled
If branch: Yes **Condition:** Flags 1, 2
Effect:

```
1. Reset control data flag (1), ack_cont flag (2) and message counters;
2. For each message recorded,
    a. If it is a special message then if it refers to this vehicle (see below),
       increment counter;
    b. else reset message flag;
    c. endif; endif; if it is a control data message, set flag 1;
    d. endif; if it is an ack_cont message, set flag 2;
    e. endif;
3. Call Controlmess;
4. Endif; go to Buschek;
```

Note. 1. At regulation level, action of in-platoon reception/transmission is to:

a. examine nature of message received; set flags and counters (using same file as messrec) indicating which messages have been received.
b. If it is a control data message, send data to controllers.
c. If it an ack_cont_dat message do no more
d. Otherwise, unless pos = 1 for a message from behind, or pos = last for a message from ahead, pass message on.

Note. 2. "Special" messages are:

	Message	***Pos***
Going forward:		
	Request-merge	1
	Request-split	1
	For-split_b_N_f	1, N
Going backward:		
	Ack_request_merge	last
	Request-mer_3_set	last
	Ack_request_split_N	(N-1)
	Invite-split-N	(N-l) if N > 2, N
	Req_spl_ch_ln_N	(N-1), N
	ForspL b_N_r	N+l, last

Name: Forprob **Fig: 1, 7**
Admitted in:N NE SA CA
Input: -
Requires: Buschek
If branch: Yes **Condition:** Flag?
Effect: 1. If flag set, call routine according to list below;
2. Endif; go to Messrec;

Flag	*Routine*
1	Insight
2,3,4	Recackprobe 1
5	Probeagain
6,7	Recackprobe2
8	Recackprobe2a
9,10,11	Recackprobenov 1
12	Novehnorep
13,14,15	Recackprobenov2
16,17,18	Reclveh
19	Setfault# 1
20	Sendackprobel
21	Sendackprobe2
22	Sendackprobenov

Name: Messrec **Fig:** 1,8
Admitted in: N NE SA CA
Input: Message list (see below)
Requires: Forprob
If branch: Yes **Condition:** Any messages received from other platoons or system?

Name: Messrec (continued)

Effect:

```
1. If busy,
2. Else examine existing flags; for each:
    a. If more than 5 cycles old, reset flag; decrement counter;
    b. endif;
3. Endif; if any messages, then for each:
    a. If emergency message from system, delete message; call appropriate
    routine; endif;
    b. If unaddressed initiating message (see below), check pos; if
    appropriate to this vehicle, call appropriate routine (see below);
    c. Delete message; decrement counter;
    d. Endif; endif; examine address, and message type; if to this platoon,
    and pos is appropriate, check forspl(x) in Buschek;
    e. If forspl(x) is set, store message;
    f. Else set message flag with cycle number;
    g. Endif; endif;
4. Endif; goto Newfault;
```

Notes. The action of a receiver is to store each message, as received, and increment a message counter. The same list and counter is used by the control data receivers, whose messages are also acted on here.

Emergency messages from system

These are obeyed by all vehicles:

Cstopcal
S topcall
Cacal
Sacal
Necal
Cutspeed
Norcal

Name: Messrec (continued)

Unaddressed initiating messages

These are noted only by vehicles in appropriate pos:

Message	***Pos***	***Routine Called***
Request-merge	1	Recreqmer
Request-merge	Last	Passon
Ack_request_merge	Last	Recackreqmerbeh
Request-mer_3_set	Last	Recmer3set
Ack_request_split_N	(N-1)	Recackrqsbeh
Request-split	1	Recrqsplit
Request_split_chnge_ln_N	N	Recreqsplchl
Request_split_chnge_ln_N	N- 1	Recreqssplchlbeh
Invite-split-N	N	Recinvsplit
Invite-split-N	N-l	Recackrqsbeh
Forspl_beh_N_f	1, (N > 1)	Leadforsplit2
Forspl_beh_N_f	N	Recbehinda
Forspl_beh_N_r	N+1	Recaheadb
Forspl_beh_N_r	Last(> N+ 1)	Lastbusy
Request-chng-ln	1	Recreqchln
Request-emerch	1	Recreqemerch
Fault-lqrobe	last	Recfault lprobe
Probe-l	last	Recprobe 1
Probe-2	last	Recprobe2
Probe-no-vehicle	last	Recprobenov
cant-go	free agent, ln# = 0	Reccantgo

Name: Newfault **Fig:** 1, 9
Admitted in: **N NE** SA CA
Input: -
Requires: Messrec
If branch: Yes **Condition:** Fault flag set? Speed? Stop ahead?

Effect:
1. If any fault flag set call Qtwofaults;
2. Endif; if maystop flag is set, ;
3. Else if speed << target, call Slowspeed;
4. Endif; endif; go to Actcalled;

A.5 BUSYCHECK
A.5.1 MERGE (Figure A.2)

Name: Busychek **Fig:** 1,2,3,4,5,6
Specification in: Sup (1)

Name: Callmerge **Fig: 2,** 10
Admitted in: N NE SA
Input: -
Requires: Mersplit (10)
If branch: No **Condition: -**
Effect: Send Req_merge; set merge(l)-flag = 1 in Busychek; reset flag in noreqm1 ; set busy; return;

Name: Contactm **Fig: 2**
Admitted in: N NE SA
Input: -
Requires: Sepmeasure and close spacing
If branch: Yes **Condition:** Flag set?
Effect:
1. Send confirm-merge; set flag = 3;
2. If flag set set stime in norepm3 = time + (increment);
3. Else set stime in norepm3 = time;
4. Endif; return;

Name: Norepm 1 **Fig: 2**
Admitted in: N NE SA
Input: -
Requires: Recnackreqm (no messages)
If branch: Yes **Condition:** Flag? long delay?
Effect:
1. If flag not set, set flag; stime = time;
2. Else if (time - stime) excessive, reset busy; call Toolong;
3. Else no action; endif; endif;
4. return;

Name: Norepm2 **Fig: 2**
Admitted in: N NE SA
Input: -
Requires: Recconmer
If branch: Yes **Condition:** Flag? long delay?
Effect:
1. If flag not set, set flag; stime = time;
2. Else if (time - stime) excessive, reset busy; call Toolong;
3. Else no action; endif; endif;
4. return;

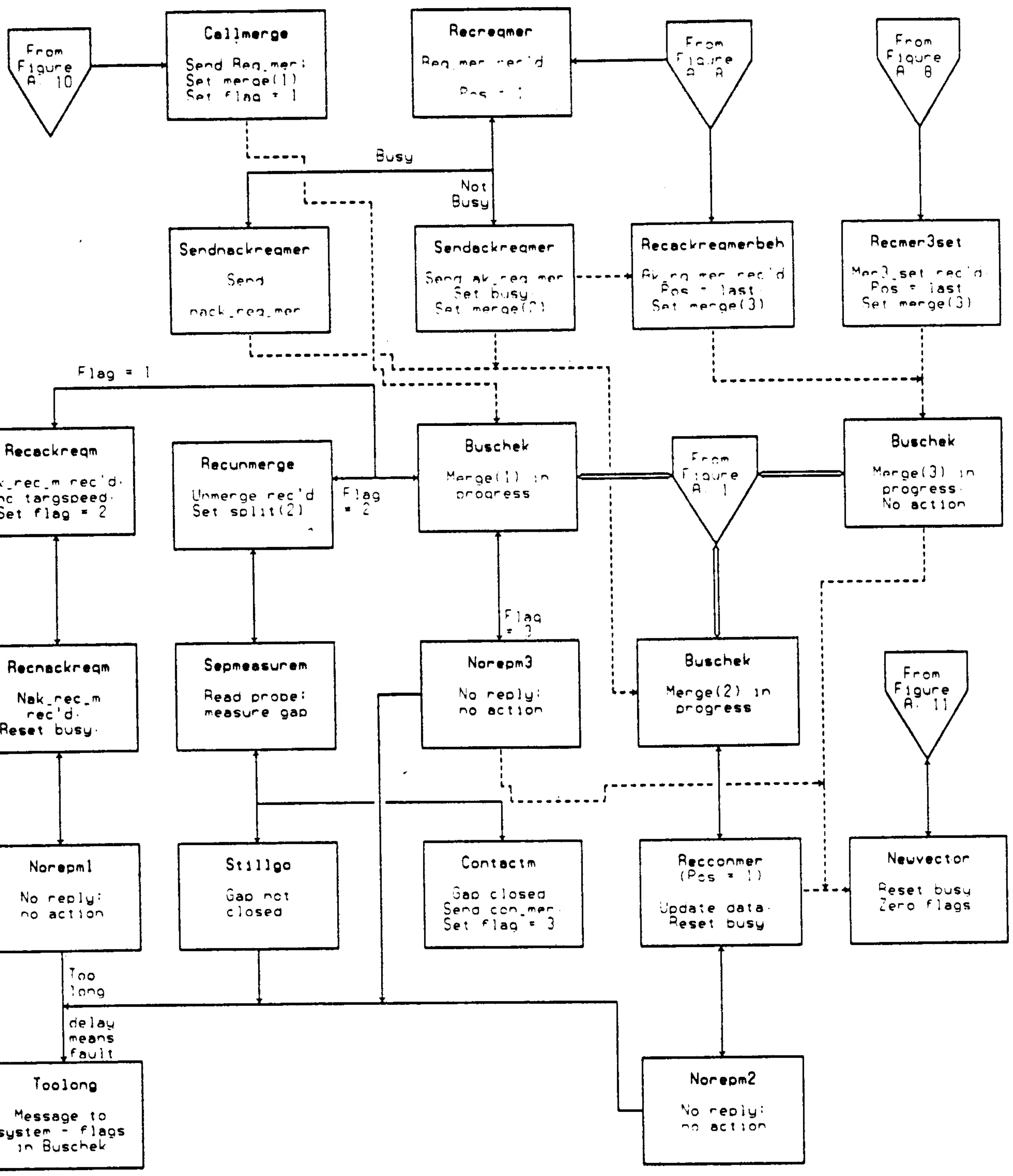


Figure A.2. Merge.

Name: Norepm3 . **Fig:** 1,2
Admitted in: **N NE** SA
Input: -
Requires: Buschek merge(l) Flag = 3
If branch: **Yes** **Condition:** (Time - stime excessive)?
Effect: 1. If (time - stime) not excessive, no action;
2. Else set split(2); reset merge(l);
3. endif; return;

Name: Recackreqm Fig: 1, 2
Admitted in: **N NE** SA
Input: Message flags
Requires: Buschek - merge(l) flag = 1
If branch: **Yes** **Condition:** Ack_request_merge rec'd?
Effect: 1. If ack_request_merge received, increment target speed; Set merge(1)-flag=2; reset message flag; reset flags in Noreqm1, Contactm; return;
2. Else call Recnackreqm; return; endif;

Name: Recackreqmerbeh **Fig: 2, 8**
Admitted in: **N NE** SA
Input: Message Ack_request_merge
Requires: Messrec pos = last
If branch: **No** **Condition:** -
Effect: Set merge(3): set busy; return;

Name: Reccon_mer **Fig:**
Admitted in: **N NE** SA
Input: -
Requires: Buschek - merge(2)
If branch: **Yes** **Condition:** Confirm-merge received?
Effect: 1. If confirm-merge received, update ownsize and transmit new control data, pltn#, etc; reset busy; reset message flag; reset flag in norepm2; reset merge(2);
2. Else call norepm2;
3. Endif; return;

Name: Recnackreqm **Fig: 2**
Admitted in: N NE SA
Input: Message flags
Requires: Recackreqm
If branch: Yes **Condition:** Nack_request_merge rec'd?
Effect:

1. If nack_request_merge received, reset busy; reset message flag; reset flag in Norepm 1; reset merge(1);
2. Else call Norepml; endif;
3. return;

Name: Recreqmer3set **Fig: 2, 8**
Admitted in: N SA
Input: Message Request-merge-3
Requires: Messrec
If branch: No **Condition: -**
Effect: Set merge(3); set busy; return;

Name: Recreq_mer **Fig: 2, 8**
Admitted in: N NE SA
Input: Message request-merge
Requires: Messrec
If branch: Yes **Condition:** Same lane? Busy?
Effect:

1. If in same lane, then
2. If not busy and no fault set, call Sendackreqmer;
3. Else call Sendnackreqmer;
4. Endif; endif;return;

Name: Recunmerge **Fig: 2**
Admitted in: N NE **SA**
Input: -
Requires: Buschek, merge(l), flag = 2
If branch: Yes **Condition:** Unmerge rec'd? Delmerge rec'd?
Effect:

1. If unmerge received, reset message flag; set split(2);
2. Else if delmerge received, reset message flag; set flag in contactm;
3. Else call sepmeasurem;
4. Endif; endif; return;

Name: Sendackreqmer . **Fig: 2, 8**
Admitted in: N NE SA
Input: -
Requires: Recreqmer and not busy
If branch: No **Condition: -**
Effect: 1. Send ack_request_merge;
2. Set busy; set merge(2); reset flag in norepm2;
3. return;

Name: Sepmeasure Fig: 1,2
Admitted in: N NE SA
Input: -
Requires: Recunmerge
If branch: Yes **Condition:** Separation <1.5*(inplat sepn)?
Effect: 1. Reads separation from veh ahead on probe.
2. If this less than (normal in-platoon separation)*1.5 call Stillgo;
3. Else call Contactm;
4. Endif; return;

Name: Sendnackreqmer **Fig: 2, 8**
Admitted in: N NE SA
Input: -
Requires: Recreqmer and busy
If branch: No **Condition: -**
Effect: Send nack_request_merge; return;

Name: S tillgo **Fig: 2**
Admitted in: N NE SA
Input: -
Requires: Sepmeasure
If branch: Yes **Condition:** Flag set?
Effect: 1. If flag not set, set flag; set stime = time;
2. Else if (time - stime) not excessive, no action
3. Else call Toolong; endif; endif;
4. return;

Name: Toolong **Fig: 2, 3**
Admitted in: N NE SA CA
Input: -
Requires: Several precursors
If branch: No **Condition: -**
Effect: Send message too-long to system (includes state vector, and flags in Buschek); reset busy; reset merge and split flags in Buschek; return;

Note. System will advise controllers - this can be a system fault, but presents no immediate danger.

A.5.2 SPLIT (Figure A.3)

Name: Busychek **Fig:** 1,2,3,4,5,6
Specification in: Sup (1)

Name: Callsplit **Fig: 3,** 10
Admitted in: N NE SA CA
Input: - ʼ
Requires: Mersplit (5) or Linkmess (5)
If branch: Yes **Condition:** Pos = 1?
Effect: 1. Sets busy;
2. If pos = 1, call invitesplit;
3. Else call reqsplit;
4. Endif; return;

Name: Invitesplit **Fig: 3,** 10
Admitted in: N NE SA CA
Input: -
Requires: Callsplit, pos = 1
If branch: No **Condition: -**
Effect: Send invite-split; Set split (1); reset flag in norepsl; update state vector; return;

Name: Newlead **Fig: 3**
Admitted in: N NE SA **CA**
Input: -
Requires: Buschek, split (2) set, flag = 2.
If branch: No **Condition: -**
Effect: Decrement target speed. set pos = 1; update state vector; Set flag = 3; Reset flags in spliton and noreps2; return;

Name: Noreps 1 **Fig: 3**
Admitted in: N NE SA CA
Input: -
Requires: Recconsplit (no messages)
If branch: Yes **Condition:** Flag? long delay?
Effect: 1. If flag not set, set flag; stime = time;
2. Else if (time - stime) excessive, call Toolong;
3. Else no action; endif; endif;
4. return;

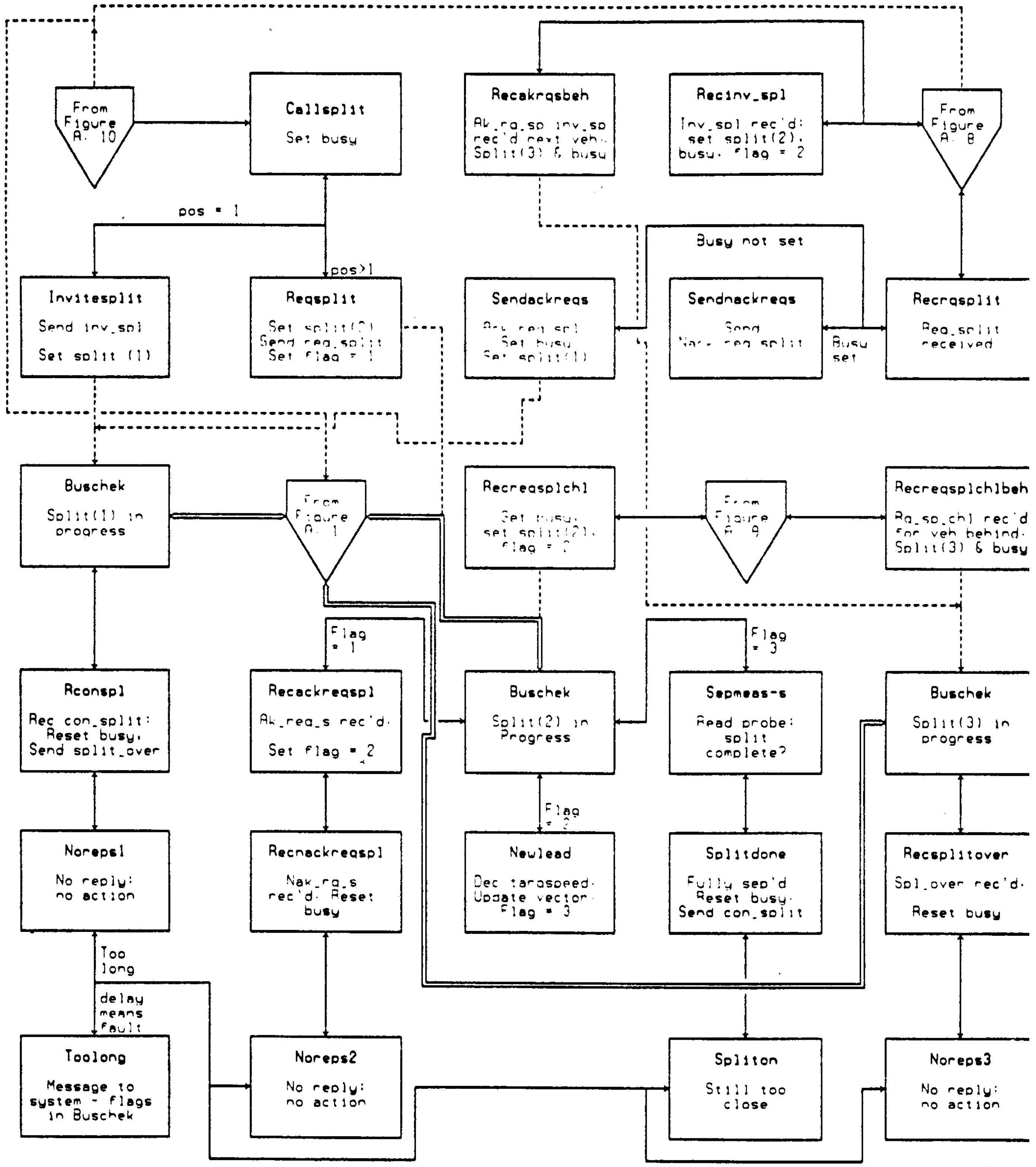


Figure A.3. Split and Split-change-lane.

Name: Noreps2 **Fig: 3**
Admitted in: N NE SA CA
Input: -
Requires: Recnackreqspl (no messages)
If branch: Yes **Condition:** Flag? long delay?
Effect: 1. If flag not set, set flag; stime = time;
2. Else if (time - stime) excessive, call Toolong;
3. Else no action; endif; endif;
4. return;

Name: Noreps3 **Fig: 3**
Admitted in: N NE SA CA
Input: -
Requires: Recsplitover (no messages)
If branch: Yes **Condition:** Flag? long delay?
Effect: 1. If flag not set, set flag; stime = time;
2. Else if (time - stime) excessive, call Toolong
3. Else no action; endif; endif;
4. return;

Name: Recackreqsplit **Fig: 3**
Admitted in: N NE SA CA
Input: -
Requires: Buschek, split(2) set, flag = 1.
If branch: Yes **Condition:** Ack_req_split received?
Effect: 1. If ack_request_split received, reset message flag; set flag = 2; reset flag in noreps2;
2. Else call Recnackreqsplit;
3. Endif; return;

Name: Recakrqsbeh **Fig: 3, 8**
Admitted in: N NE SA CA
Input: Message Ack_request_split or invite-split to vehicle behind
Requires: Messrec
If branch: No **Condition: -**
Effect: Reset message flag; set busy; set split(3); reset flag in noreps3; return;

Name: Reconsplit **Fig: 3**
Admitted in: N NE SA CA
Input: -
Requires: Buschek, split(1) set
If branch: Yes **Condition:** confirm-split received?
Effect: 1. If confirm-split received, reset message flag; reset busy; send split-over; reset flag in norepsl; reset split(l);
2. Else call norepsl ;
3. Endif; return;

Name: Reqsplit Fig: 3, 10
Admitted in: N NE SA CA
Input: -
Requires: Callsplit, pos > 1
If branch: No **Condition: -**
Effect: Send request-split; set split(2); set flag =1; reset flag in noreps2; return;

Name: Recinv_spl **Fig: 3, 8**
Admitted in: N NE SA **CA**
Input: Message invite-split
Requires: Messrec
If branch: No **Condition: -**
Effect: Reset message flag; set busy; set split(2) - flag = 2; return;

Name: Recnackreqs **Fig: 3**
Admitted in: N NE SA CA
Input: -
Requires: Recackreqs, ack not received
If branch: Yes **Condition:** Nack_request_split received?
Effect: 1. If nack_request_split received, reset busy; reset message flag; reset split(2); reset flag in noreps2;
2. Else call noreps2;
3. endif; return;

Name: Recreqsplit **Fig: 3, 8**
Admitted in: N NE SA CA
Input: Message request-split
Requires: Messrec and pos = 1;
If branch: Yes **Condition:** Same lane? Busy?
Effect: 1. If same lane, then
2. If busy or fault set, call Sendnackreqs;
3. Else call Sendackreqs;
4. Endif; endif; return;

Name: Recreqsplchl . **Fig: 3, 8**
Admitted in: N SA
Input: Message Req_spl_change_lane
Requires: Messrec
If branch: No **Condition: -**
Effect: Set split(2); set split(2)-flag = 2; reset flag in noreps2; set busy; return;

Name: Recreqsplchlbeh **Fig: 3, 8**
Admitted in: N SA
Input: Message request-split-change-lane rec'd for vehicle behind
Requires: Messrec
If branch: No **Condition: -**
Effect: Set split(3); set busy; reset flag in noreps3; return;

Name: Recsplitover **Fig: 3**
Admitted in: N NE SA CA
Input: -
Requires: Buschek, split(3).
If branch: Yes **Condition:** split-over received?
Effect:
1. If split-over received, reset message flag; reset busy; reset flag on noreps3; reset split(3);
2. Else call noreps3;
3. endif; return;

Name: Sendackreqs **Fig: 3, 8**
Admitted in: N NE SA CA
Input: -
Requires: Recreqsplit
If branch: No **Condition: -**
Effect: Send ack_request_split; set busy; set split(l); reset flag in norepsl; update state vector; return

Name: epmeasures **Fig: 3**
Admitted in: N NE SA CA
Input:
Requires: Buschek, split(2)-flag = 3.
If branch: No **Condition: -**
Effect: Read from probe separation from vehicle ahead; pass this in calling Splitdone; return;

Name: Sendnackreqs **Fig: 3, 8**
Admitted in: N NE SA CA
Input: -
Requires: Recreqsplit and busy set
If branch: No **Condition: -**
Effect: Send nack_request_split; return

Name: Spliton **Fig: 3**
Admitted in: N NE SA CA
Input: -
Requires: Splitdone (no messages)
If branch: Yes **Condition:** Flag? long delay?
Effect: 1. If flag not set, set flag; stime = time;

```
    2. Else if (time - stime) excessive, call Toolong;
    3. Else no action; endif; endif;
    4. return;
```

Name: Splitdone **Fig: 3**
Admitted in: N NE SA CA
Input: Distance from veh ahead (= d)
Requires: Sepmeasures,
If branch: Yes **Condition:** d < platoon spacing?
Effect: 1. If d < platoon spacing, call spliton;

```
    2. Else send confirm-split; reset busy; reset flag in Spliton; reset split(2);
    3. endif; return;
```

Name: Toolong **Fig: 2, 3**
Specification in: Merge (2).

AS.3 CHANGELANE (Figure A.4)

Name: Busychek **Fig:** 1,2,3,4,5,6
Specification in: Sup (1)

Name: Callchangelane **Fig: 4,** 10
Admitted in: N NE SA CA
Input: -
Requires: Linkmess (10)
If branch: No **Condition: -**
Effect: Set busy; set chng(1); set flag = 1; call Reqchngln; return;

Name: Chltomer2 **Fig: 4**
Admitted in: N SA
Input: -
Requires: Sendconchl2
If branch: No **Condition: -**
Effect: Reset chng(1); set merge(2); return;

Name: Compposns **Fig: 4**
Admitted in: N SA
Input: -
Requires: Buschek, chng(2) and flag = 4
If branch: Yes **Condition:** Results of calculation
Effect:
1. From pos, speed of entering car, own speed, ownsize, pos and data sent, determine change-lane strategy:
2. If veh is to join behind, call Sendreqdecel; flag = 5;
3. Else if in centre, call Sendreqsplchll; flag = 5;
4. Else ahead, call Dropback; flag = 6;
5. Endif; endif; return;

Name: Dropback **Fig: 4**
Admitted in: N SA
Input: -
Requires: Compposns
If branch: No **Condition:**
Effect:
1. Determines position and velocity of both vehicles so that enterer will arrive at gate at head of platoon;
2. Records targets, stores them; flag = 6; return;

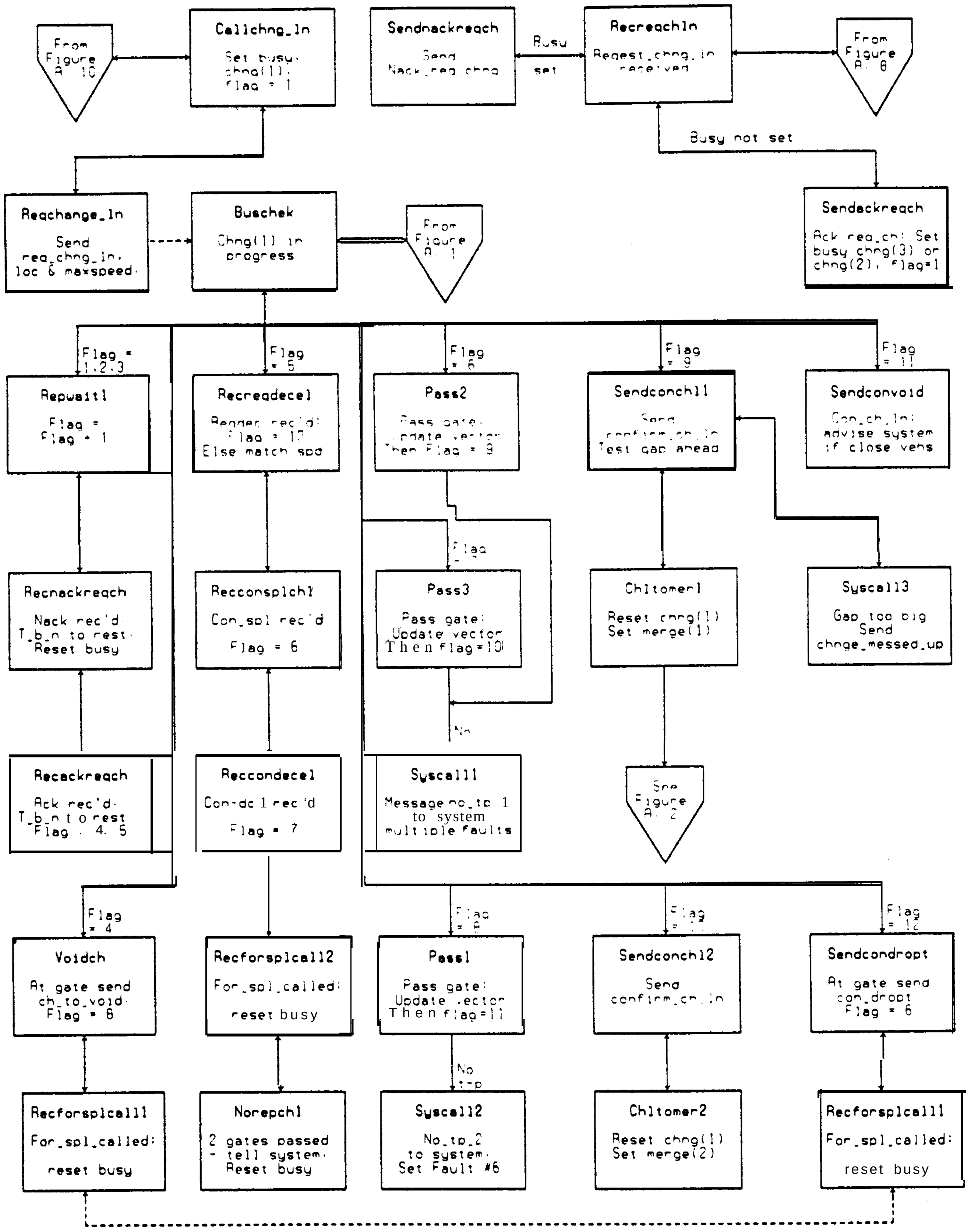
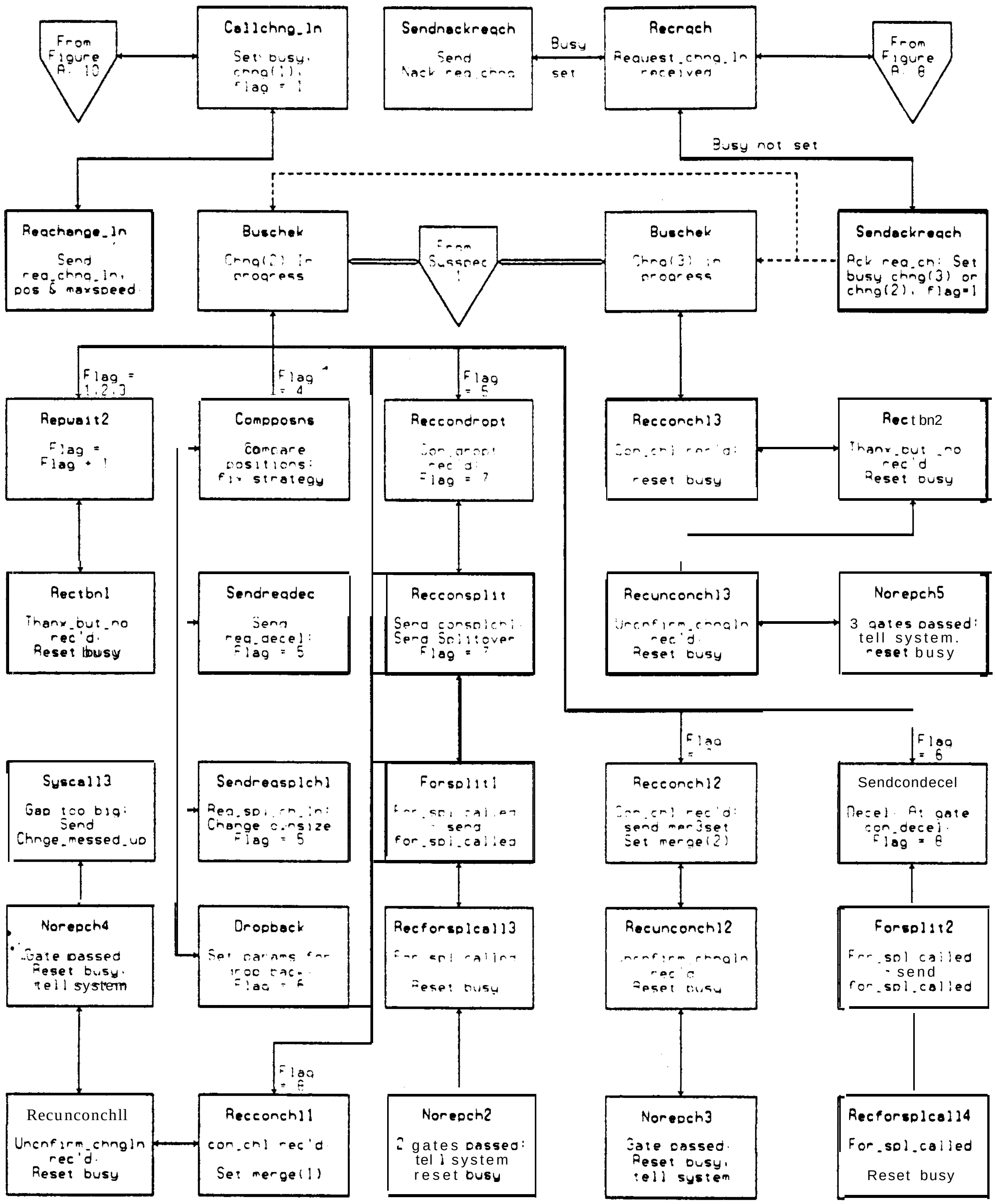

From Figure A. 10
Callchng_ln
Set busy chng(1); flag = 1
Sendnackreach
Send Nack_req_chng
Busy set
Recreachln
Reqest_chng_ln received
From Figure A. 8
Busy not set
Sendackreach
Ack req_ch: Set busy chng(3) or chng(2), flag=1
Reqchange_ln
Send req_chng_ln, loc & maxspeed
Buschek
Chng(1) in progress
From Figure A. 1
Flag = 1,2,3
Flag = 5
Flag = 6
Flag = 9
Flag = 11
Repwait1
Flag = Flag + 1
Recreadecel
Readec rec'd: flag = 10 Else match spd
Pass2
Pass gate: Update vector Then Flag = 9
Sendconchl1
Send confirm_ch_ln Test gap ahead
Sendconvoid
Con_ch_ln: advise system if close vehs
Flag
Recnackreach
Nack rec'd: T_b_n to rest: Reset busy
Recconsplchl
Con_spl rec'd Flag = 6
Pass3
Pass gate: Update vector Then flag=10
Chltomerl
Reset chng(1) Set merge(1)
Syscall3
Gap too big Send chnge_messed_up
No
Recackreach
Ack rec'd: T_b_n to rest Flag . 4. 5
Reccondecel
Con-dc 1 rec 'd Flag = 7
Syscall1
Message no_tp 1 to system multiple faults
See Figure A. 2
Flag = 4
Flag = 8
Flag = 12
Flag = 12
Voidch
At gate send ch_to_void Flag = 8
Recforsplcall2
For_spl_called: reset busy
Pass1
Pass gate: Update vector Then flag=11
Sendconchl2
Send confirm_ch_ln
Sendcondropt
At gate send con_dropt Flag = 6
No t-p
Recforsplcall1
For_spl_called: reset busy
Norepchl
2 gates passed - tell system. Reset busy
Syscall2
No_tp_2 to system. Set Fault #6
Chltomer2
Reset chng(1) Set merge(2)
Recforsplcall1
For_spl_called: reset busy

Figure A.4. Change Lane (initiator).
(t-p = turning point)



From Figure A. 10
Callchng_ln
Set busy, chng(1), Flag = 1
Sendnackreqch
Send Nack req_chng
Busy set
Recrqch
Request_chng_ln received
From Figure A. 8
Busy not set
Reqchange_ln
Send req_chng_ln, pos & maxspeed
Buschek
Chng(2) in progress
From Susspec 1
Buschek
Chng(3) in progress
Sendackreqch
Ack req_ch: Set busy chng(3) or chng(2), Flag=1
Flag = 1,2,3
Flag = 4
Flag = 5
Repwait2
Flag = Flag + 1
Comppospns
Compare positions: fix strategy
Reccondropt
Con_dropt rec'd: Flag = 7
Recconch13
Con_chl rec'd: reset busy
Rectbn2
Thanx_but_no rec'd Reset busy
Rectbnl
Thanx_but_no rec'd. Reset busy
Sendreqdec
Send req_decel: Flag = 5
Recconsplit
Send consplchl: Send Splitover Flag = 7
Recunconch13
Unconfirm_chngln rec'd. Reset busy
Norepch5
3 gates passed: tell system. reset busy
Flag = 7
Flag = 6
Syscall3
Gap too big: Send Chnge_messed_up
Sendreqsplchl
Req_spl_ch_ln: Change cunsize Flag = 5
Forsplit1
For_spl_called - send for_spl_called
Recconch12
Con_chl rec'd: send mer3set Set merge(2)
Sendcondecel
Decel: At gate con_decel. Flag = 8
Norepch4
Gate passed Reset busy. tell system
Dropback
Set params for drop back. Flag = 6
Recforsplcall13
For_spl_called Reset busy
Recunconch12
Unconfirm_chngln rec'd Reset busy
Forsplit2
For_spl called - send for_spl_called
Flag = 8
Recunconchll
Unconfirm_chngln rec'd. Reset busy
Recconch11
con_chl rec'd Set merge(1)
Norepch2
2 gates passed: tell system reset busy
Norepch3
Gate passed. Reset busy, tell system
Recforsplcall14
For_spl_called Reset busy

Figure A.4a. Change Lane (recipient).
(t-p = turning point)

Name: Forsplitl **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Recconsplit
If branch: Yes **Condition:** Forced split called? sect in NE?
Effect: 1. **If** forced split called (ie forspl(1) in buschek set), send forced-split-called; **reset chng(2);**
2. Else if section is now in NE **or CA,** send-forced-split- called; reset chng(2); reset busy;
3. Endif; call Recforsplcall3; return;

Name: Forsplit2 **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Sendcondecel
If branch: Yes **Condition:** Forced split called? sect in NE?
Effect: 1. If forced split called (ie forspl(1) in buschek set), send forced-split-called; reset chng(2);
2. Else if section is now in NE or CA, send-forced-split- called; reset chng(2); reset busy;
3. Endif; call Recforsplcall4; return;

Name: Norepch 1 **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Recforsplcall2
If branch: Yes **Condition:** Gate passed? Counter > 1?
Effect: 1. If gate signal received, then if signal indicates section of receiver is in NE, reset busy;
2. Else counter + = 1; if counter > 1, then reset busy; advise system - no confirmation from platoon . . . ;
3. Endif; endif; endif; return;

Name: Norepch2 **Fig: 4**
Admitted in: N SA
Input: -
Requires: Recforsplcall3
If branch: Yes **Condition:** Gate present? Counter > 1? Flag?

Effect:
```
1. If gate present, then
    2. If flag reset, set flag; counter + = 1;
    3. Endif; Else if counter > 1, reset busy; reset chng(2); advise system;
    4. Else if flag set, reset flag;
    5. Endif; endif; endif; return;
```

Name: Norepch3 **Fig: 4**
Admitted in: N SA
Input: -
Requires: Recunconchl2
If branch: Yes **Condition:** Gate present? Counter > O? Flag?

Effect: 1. If gate present, then
2. If ‘flag reset, set flag; counter + = 1;
3. Endif; Else if counter > 1, reset busy; reset chng(2); advise system;
4. Else if flag set, reset flag;
5. Endif; endif; endif; return;

Name: Norepch4 **Fig: 4**
Admitted in: N SA
Input: -
Requires: Recunconchll
If branch: Yes **Condition:** Gate present?
Effect: 1. If gate present, ;
2. Else reset busy; reset chng(2); advise system;
3. Endif; return;

Name: Norepch5 **Fig: 4**
Admitted in: N SA
Input: -
Requires: Recunconchl3
If branch: Yes **Condition:** Gate present? Counter > 2? Flag?

Effect: 1. If gate present, then
2. If flag reset, set flag; counter + = 1;
3. Endif; Else if counter > 2, reset busy; reset chng(2); advise system;
4. Else if flag set, reset flag;
5. Endif; endif; endif; return;

Name: Pass1 **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Buschek, chng(1) and flag = 8
If branch: Yes **Condition:** Gate detected? Turning point? Flag 1, Flag 2 set?

Effect:
```
1. If gate detected, then
    2. If flag 1 set, then
    3. If t-p detected, turn to pass gate; update state vector; set flag 2;
    4. Endif; else set flag 1;
    5. Endif; else if flag 2 set; flag (buschek) = 11;
    6. Else if flag 1 set, call Syscall2; reset busy;
    7. Endif; endif; endif; return;
```

Name: Pass2 **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Buschek, chng(1) and flag = 6
If branch: Yes **Condition:** Gate detected? Turning point? Flag 1, Flag 2 set?

Effect:
```
1. If gate detected, then
    2. If flag 1 set, then
    3. If t-p detected, turn to pass gate; update state vector; set flag 2;
    4. Endif; else set flag 1;
    5. Endif; else if flag 2 set; flag (buschek) = 9;
    6. Else if flag 1 set, call Syscalll; reset busy;
    7. Endif; endif; endif; return;
```

Name: Pass3 **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Buschek, chng(1) and flag = 7
If branch: Yes **Condition:** Gate detected? Turning point? Flag 1, Flag 2 set?

Effect:
```
1. If gate detected, then
    2. If flag 1 set, then
    3. If t-p detected, turn to pass gate; update state vector; set flag 2;
    4. Endif; else set flag 1;
    5. Endif; else if flag 2 set; flag (buschek) = 10;
    6. Else if flag 1 set, call Syscalll; reset busy;
    7. Endif; endif; endif; return;
```

Name: Recackreqchl **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Recnackreqchl
If branch: Yes **Condition:** Ack_req_chng_ln received?
Effect:
1. Send thanx_but_no to all irrelevant replies;
2. If relevant reply received from adjacent lane, store stated speed; set flag =4; reset counter in norepchl;
3. Else if reply only from next lane, store stated speed;
4. Endif; endif; reset all message flags; return;

Name: Reccondecel **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Recconsplchl
If branch: Yes **Condition:** Confirm-decel received?
Effect: 1. If confirm-decel received,
2. Reset message flag; if in position,
3. Flag = 7; reset both flags in Pass3;
4. Else set fault #11;
5. Endif; else call Recforsplcalll ;
6. Endif; return;

Name: Recconchll **Fig: 4**
Admitted in: N SA
Input: -
Requires: Buschek, chng(2) and flag = 7
If branch: Yes **Condition:** Confirm-change-ln received?
Effect: 1
1. If confirm-change-lane received, reset chng(2); set merge(l); if gap ahead is too big call Syscall3;
2. Endif; else call Recunconchll;
3. Endif; return;

Name: Recconchl2 **Fig: 4**
Admitted in: N SA
Input: -
Requires: Buschek, chng(2) and flag = 7
If branch: Yes **Condition:** Confirm-change-ln received?
Effect:
1. If confirm-change-lane received, send mer3set; reset chng(2); set merge(2);
2. Else call Recunconchl2;
3. Endif; return;

Name: Recconchl3 **Fig: 4**
Admitted in: N SA
Input: -
Requires: Buschek and chng(3)
If branch: Yes **Condition:** Confirm-change-lane received?
Effect:

```
1. If confirm-change-lane received, reset busy; reset chng(3); reset message marker;
2. Else call Rectbn2;
3. Endif; return;
```

Name: Reccondropt **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Buschek, Chng(2), flag = 5
If branch: Yes **Condition:** Confirm-dropt received?
Effect:

```
1. If confirm-dropt received, then if in position,
    2. set flag = 7; reset counter in Norepch3; reset message flag;
    3. Else send forsplit-called;
    4. Endif; else adjust speed to meet target; call Recconsplit;
    5. Endif; return;
```

Name: Recconsplit **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Reccondropt
If branch: Yes **Condition:** Confirm-split received?
Effect:

```
1. If confirm-split received,
    2. If near gate, reset message flag; send split-over;
    3. If in position, send confirm-split-change-lane; reset counter in Norepch3; flag=7;
    4. Else send for-split-called;
    5. Endif; else continue to achieve position; call Forsplitl;
    6. Endif; else call Forsplitl;
    7. Endif; return;
```

Name: Recconsplitchl . **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Recreqdecel
If branch: Yes **Condition:** Confirm-split-chng-In received?
Effect: 1. If confirm_split_change_lane received,
2. Reset message flag; if in position,
3. Flag = 6; reset both flags in pass2;
4. Else set fault #11;
5. Endif; else call Reccondecel;
6. Endif; return;

Name: Rectbn 1 **Fig: 4**
Admitted in: N SA
Input: -
Requires: Repwait2
If branch: Yes **Condition:** Thanx_but_no received:
Effect: 1. If thanx-but-no received, reset busy; reset message marker; reset chng(2);
2. Endif; return;

Name: Rectbn2 **Fig: 4**
Admitted in: N SA
Input: -
Requires: Recconchl3
If branch: Yes **Condition:** Thanx_but_no received:
Effect: 1. If thanx-but-no received, reset busy; reset message marker; reset chng(3);
2. Else call Recunconchl3;
3. Endif; return;

Name: Repwait 1 **Fig: 4**
Admitted in: N NE **SA CA**
Input: -
Requires: Buschek, chnge(1), flag = 1, 2, or 3.
If branch: No **Condition: -**
Effect: Call Recnackch; flag + = 1; return;

Name: Repwait2 **Fig: 4**
Admitted in: N SA
Input: -
Requires: Buschek, chnge(2), flag = 1, 2, or 3.
If branch: No **Condition: -**
Effect: Call Rectbnl; flag + = 1; return;

Name: Recforsplcalll **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Voidch, Sendcondropt
If branch: Yes **Condition:** Forced-split-called received?
Effect: 1. If forced_split_called received, reset busy; reset message flag;
2. endif; return;

Name: Recforsplcall2 **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Reccondecel
If branch: Yes **Condition:** Forced_split_called received?
Effect: 1. If forced_split_called received, reset message flag; reset busy;
2. Else call Norepchl;
3. endif; return

Name: Recforsplcall3 **Fig: 4**
Admitted in: N NE SA **CA**
Input: -
Requires: Forsplit 1
If branch: Yes **Condition:** Forced-split-called received?
Effect: 1. If forced_split_called received, reset busy; reset message flag;
2. Endif; call norepch2; return;

Name: Recforsplcall4 **Fig: 4**
Admitted in: N NE SA **CA**
Input: -
Requires: Forsplit2
If branch: Yes **Condition:** Forced_split_called received?
Effect: 1. If forced-split-called received, reset busy; reset message flag;
2. endif; return;

Name: Recnackreqchl **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Repwait
If branch: Yes **Condition:** Flag = 3? Nack_rq_chng_ln received?

Effect:
1. If flag = 3, then
2. If veh in adjacent lane has sent nack_request_chng_ln, send thanx-but-no to all those who have replied, except the nack_er; reset busy;
3. Else if no replies from adjacent lane, and veh in lane two away has sent nack_request_change-lane send thanx-but-no to all others; reset busy; reset all message flags;
4. Else call Recreqackchl;
5. Endif; endif; endif; return;

Name: Reqchngln **Fig: 4,** 10
Admitted in: N NE SA CA
Input: -
Requires: Callchangelane
If branch: No **Condition: -**
Effect: Send req_chng_lane, including loc and maxspeed; return;

Name: Recreqchngln . **Fig: 4, 8**
Admitted in: N NE SA CA S C
Input: Loc and maxspeed of sender.
Requires: Messrec
If branch: Yes **Condition:** Within 2 lanes on side of change? Busy? in N or **SA?**
Effect: 1. If in adjacent or next lane on side of change, and within platoon spacing of ' loc, then
2. If "busy" not set, and mode is N or SA, and no fault set, call Sendackreqchln with data received;
3. Else call Sendnackreqchln;
4. Endif; endif; return;

Name: Recreqdecel; **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Buschek, chng(1), flag = 5.
If branch: Yes **Condition:** Request-decel received?
Effect: 1. If request-decel received, flag = 11; store target time of arrival at gate; reset message flag;
2. Else call Recconspl; set speed to stated speed of requester;
3. Endif; return;

Name: Recunconchll **Fig: 4**
Admitted in: N SA
Input: -
Requires: Recconchl2
If branch: Yes **Condition:** Unconfirm_change_ln received?
Effect: 1. If unconfirm change-lane received, reset chng(2); reset busy; reset message flag;
2. Else call Norepch4;
3. Endif; return;

Name: Recunconchl2 **Fig: 4**
Admitted in: N SA
Input: -
Requires: Recconchl2
If branch: Yes **Condition:** Unconfirm_change_ln received?
Effect: 1. If unconfirm change-lane received, reset chng(2); reset busy; reset message flag;
2. Else call Norepch3;
3. Endif; return;

Name: Recunconchl3 **Fig: 4**
Admitted in: N SA
Input: -
Requires: Tectbn2
If branch: Yes **Condition:** Unconfirm_chng_ln received?
Effect: 1. If unconfirm_change_lane received, reset busy; reset message flag; reset chng(3);
2. Else call norepch5;
3. Endif; return;

Name: Sendackreqchln **Fig: 4, 8**
Admitted in: N SA
Input: loc and maxspeed of requester
Requires: Recreqchngln, not busy and right mode
If branch: Yes **Condition:** Lane adjacent? next to this?
Effect: 1. Set new speed - minimum of own and requester's maxspeed;
2. Send Ack_request_change_lane with lane and new speed; set busy; store data sent and received;
3. If in adjacent lane, set chng(2); set flag = 1;
4. Else set chng(3); reset counter and flag in Norepch5;
5. Endif; return;

Name: Sendconchln 1 **Fig: 4**
Admitted in: N SA
Input: -
Requires: Buschek, chng(1) and flag = 9
If branch: Yes **Condition:** Gap too big?
Effect: 1. Send confirm-change-lane to all participating agents;
2. If gap ahead too big; call Syscall3;
3. Endif; call Chltomerl; return;

Name: Sendconchln2 **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Buschek, chng(1) and flag = 10
If branch: No **Condition:** -
Effect: Send confirm-change-lane to all participating agents; call chltomerl; return;

Name: Sendcondecel . **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Buschek, chng(2), flag = 6
If branch: Yes **Condition:** Gate near?
Effect: 1. Adjust speed to reach gate at target time;
2. If gate near, then if in position,
3. Send confirm-decel; flag = 8;
4. Else send forsplit-called;
5. Endif; else call forsplit2;
6. Endif; return;

Name: Sendcondropt **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Buschek, chng(1) and flag = 12
If branch: Yes **Condition:** Gate reached?
Effect: 1. Adapt speed to reach gate at target time;
2. If gate reached,
3. If in position, flag = 6; send confirm-dropt; reset flags in pass2;
4. Else send forsplit-called;
5. Endif; else call Recforsplcalll ;
6. Endif; return;

Name: Sendconvoid **Fig: 4**
Admitted in: N SA
Input: Reading from probe
Requires: Buschek, chng(1) and flag = 11
If branch: Yes **Condition:** Vehs in platoon spacing?
Effect: 1. Send confirm-change-lane to any participating agent;
2. If vehicles ahead in platoon spacing detected, advise system;
3. Endif; reset busy; reset chng(1); return;

Name: Sendnackreqchln **Fig: 4, 8**
Admitted in: N NE SA **CA S C Q**
Input: -
Requires: Recreqchln and (busy or wrong mode)
If branch: No **Condition: -**
Effect: Send nack_request_change_lane; return;

Name: Sendreqdecel **Fig: 4**
Admitted in: N SA
Input: -

Requires: Compposns
If branch: No **Condition: -**
Effect:
1. Determines position and velocity of both vehicles so that enterer will arrive at gate at back of platoon;
2. Records targets, stores them; sends request-decel with this data;
3. Resets counter, flag in norepch2; flag = 5;
4. return;

Name: Sendreqsplchl **Fig: 3, 4**
Admitted in: N SA
Input: -
Requires: Compposns
If branch: No **Condition: -**
Effect:
1. Determines position and velocity of both vehicles so that enterer will arrive at gate behind vehicle (N-l) of platoon;
2. Records targets, stores them; sends request-split-change-lane to vehicle N;
3. Resets counter, flag in norepch2; flag = 5;
4. return;

Name: S yscalll **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Pass2, pass3, and no t-p
If branch: No **Condition: -**
Effect: Send unconfirm_change_lane; Send to system no_tp_1; return;

Name: Syscall2 **Fig: 4**
Admitted in: N NE SA CA
Input: -
Requires: Pass1 , and no t-p
If branch: No **Condition: -**
Effect: Send unconfirm-change-lane; send to system no_tp_2; set fault #6; return;

Name: Syscall3 **Fig:** 4
Admitted in: N SA
Input: -
Requires: Sendconchll or Recconchll and big gap
If branch: No **Condition:** -
Effect: Send change-messed-up (with ID's of participants); return;

Name: Voidchange **Fig:** 4
Admitted in: N NE SA CA
Input: -
Requires: Buschek, chng(1), flag = 4
If branch: Yes **Condition:** Gate near?
Effect:

1. If gate near (as determined by loc and its broadcast of its loc), send change-to-void; flag = 8; reset both flags in Passl;
2. Else call Recforsplcall1;
3. Endif; return;

AS.4 FORCED-SPLIT (Figure A.5)

Name: Buslead 1 Fig: 5, 10, 11
Admitted in: **N NE** SA CA
Input: -
Requires: Fsl set, (pos = 1) and busy
If branch: Yes **Condition:** multiple
Effect: 1. Action according to flag set in buschek:

Merge(l): set forsplit(4); store data; set active flag; break;
Merge(2): reset merge(2); set forsplit(1); set time in Norepforsl; break;
Split(1): reset split(l); set forsplit(1); set time in Norepforsl; break;
Split(2); set forsplit(5); store data; set active flag; break;
Chng(2): set forsplit(1); set time in Norepforsl; break;
Chng(3): store data; set active flag; set forsplit(1); set time in Norepforsl; break;
Emerch(x), x = 2,3,4,5 set forsplit(1); set time in Norepforsl; advise system; break;
Forspl(x), x = 1,6: Call stop mode; advise system;

2. endifs; return;

Name: Buslead2 **Fig: 5, 8**
Admitted in: **N NE** SA CA
Input: -
Requires: Leaqforsplitl and busy
If branch: Yes **Condition:** multiple
Effect: 1. Action according to flag set in buschek;

Merge(l): set forsplit(4); store data; set active flag; break;
Merge(2): reset merge(2); set forsplit(1); set time in Norepforsl; break;
Split(l): reset split(l); set forsplit(1); set time in Norepforsl; break;
Split(2); set forsplit(5); store data; set active flag; break;
Chng(2): set forsplit(1); set time in Norepforsl; break;
Chng(3): store data; set active flag; set forsplit(1); set time in Norepforsl; break;
Emerch(x), x = 2,3,4,5 set forsplit(1); set time in Norepforsl; advise system; break;
Forspl(x), x = 1,6: Call stop mode; advise system;

2. endifs; return;

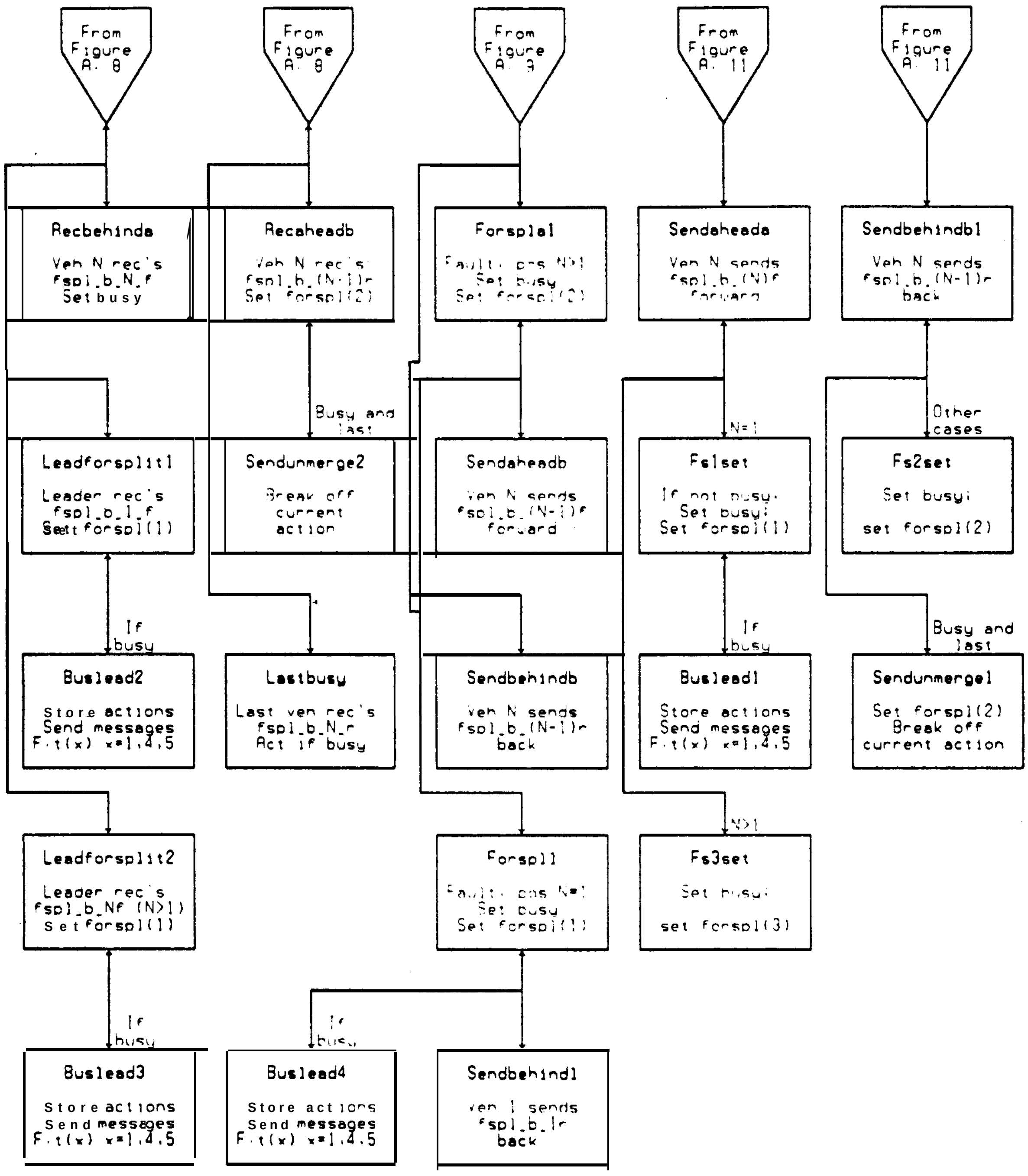


Figure AS. Forced Split (initiation).

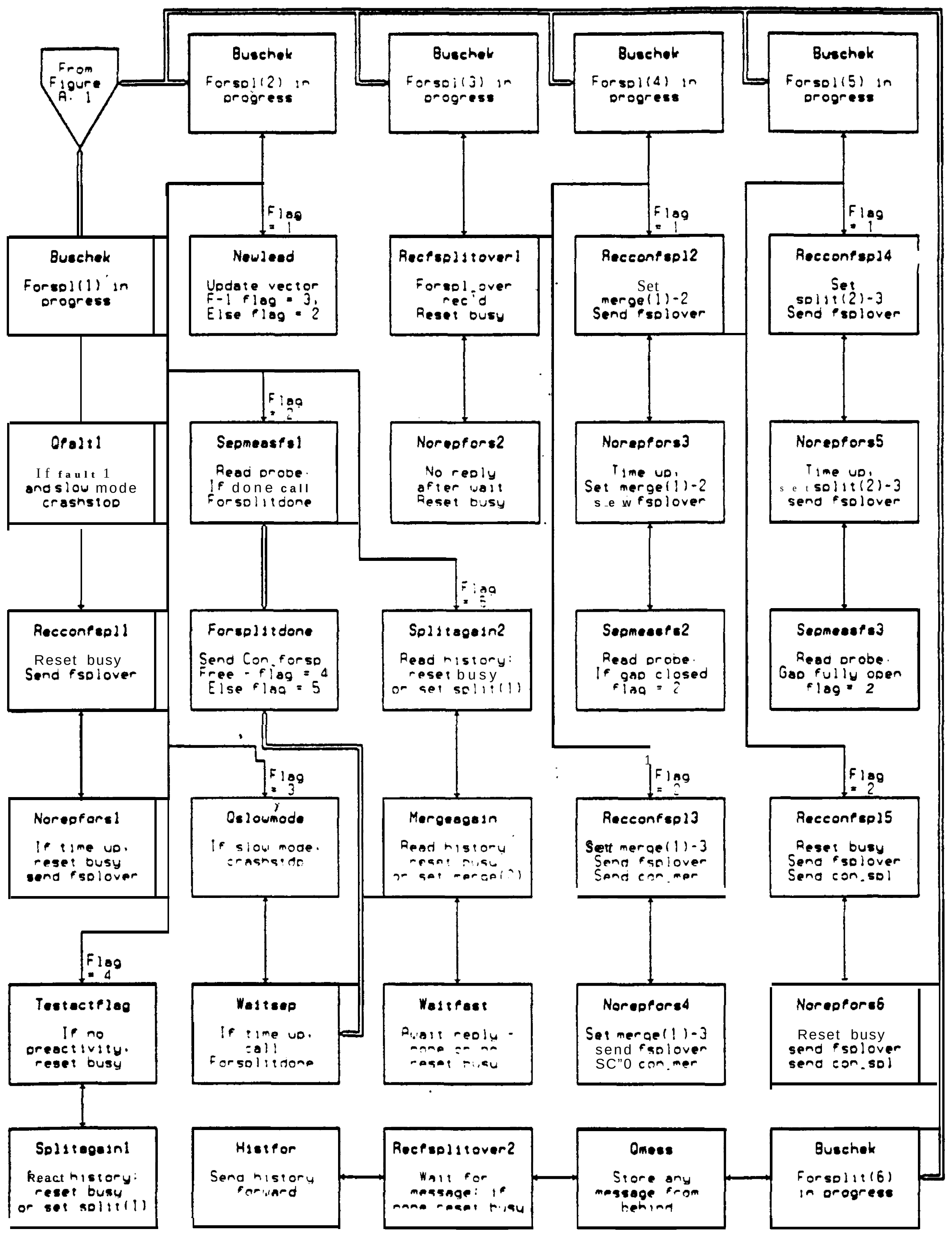

From Figure A. 1
Buschek Forspl(2) in progress
Buschek Forspl(3) in progress
Buschek Forspl(4) in progress
Buschek Forspl(5) in progress
Buschek Forspl(1) in progress
Flag = 1
Newlead Update vector F-1 flag = 3, Else flag = 2
Recfsplitover1 Forspl over rec'd Reset busy
Flag = 1
Recconfspl2 Set merge(1)-2 Send fsplover
Flag = 1
Recconfspl4 Set split(2)-3 Send fsplover
Flag = 2
Qfalt1 If fault 1 and slow mode crashstop
Sepmeasfs1 Read probe. If done call Forsplitdone
Norepfors2 No reply after wait Reset busy
Norepfors3 Time up, Set merge(1)-2 send fsplover
Norepfors5 Time up, set split(2)-3 send fsplover
Flag = 5
Recconfspl1 Reset busy Send fsplover
Forsplitdone Send Con_forsp Free - flag = 4 Else flag = 5
Splitagain2 Read history: reset busy or set split(1)
Sepmeasfs2 Read probe. If gap closed flag = 2
Sepmeasfs3 Read probe. Gap fully open flag = 2
Flag = 3
Flag = 2
Flag = 2
Norepfors1 If time up, reset busy send fsplover
Qslowmode If slow mode, crashstop
Mergeagain Read history reset busy or set merge(2)
Recconfspl3 Set merge(1)-3 Send fsplover Send con_mer
Recconfspl5 Reset busy Send fsplover Send con_spl
Flag = 4
Testactflag If no preactivity, reset busy
Waitsep If time up, call Forsplitdone
Waitfast Await reply - none or no reset busy
Norepfors4 Set merge(1)-3 send fsplover send con_mer
Norepfors6 Reset busy send fsplover send con_spl
Splitagain1 React history: reset busy or set split(1)
Histfor Send history forward
Recfsplitover2 Wait for message: if none reset busy
Qmess Store any message from behind
Buschek Forsplit(6) in progress

Figure A.5a. Forced Split (implementation).

Name: Buslead3 . **Fig: 5, 8**
Admitted in: N NE SA CA
Input: -
Requires: Leadforsplit2 and busy
If branch: Yes **Condition:** multiple
Effect: 1. Action according to flag set in buschek;
Merge(l): set forsplit(4); store data; set active flag; break;
Merge(2): reset merge(2); set forsplit(1); set time in Norepforsl; break;
Split(l): reset split(l); set forsplit(1); set time in Norepforsl; break;
Split(2); set forsplit(5); store data; set active flag; break;
Chng(2): set forsplit(1); set time in Norepforsl; break;
Chng(3): store data; set active flag; set forsplit(1); set time in Norepforsl; break;
Emerch(x), x = 2,3,4,5 set forsplit(1); set time in Norepforsl; advise system; break;
Forspl(x), x = 1,6: Call stop mode; advise system;
2. endifs; return;

Name: Buslead4 **Fig: 5,** 10
Admitted in: N NE SA CA
Input: -
Requires: Forspll and busy
If branch: Yes **Condition:** multiple
Effect: 1. Action according to flag set in buschek;
Merge(l): set forsplit(4); store data; set active flag; break;
Merge(2): reset merge(2); set forsplit(1); set time in Norepforsl; break;
Split(l): reset split(l); set forsplit(1); set time in Norepforsl; break;
Split(2); set forsplit(5); store data; set active flag; break;
Chng(2): set forsplit(1); set time in Norepforsl; break;
Chng(3): store data; set active flag; set forsplit(1); set time in Norepforsl; break;
Emerch(x), x = 2,3,4,5 set forsplit(1); set time in Norepforsl; advise system; break;
Forspl(x), x = 1,6: Call stop mode; advise system;
2. endifs; return;

Name: Busychek Fig: 1,2,3,4,5,6
Specification in: Sup (1)

Name: Forspla **Fig: 5,** 10
Admitted in: N NE SA CA
Input: -
Requires: Actcalled
If branch: No **Condition: -**
Effect: Set busy; set forspl(2); set flag = 1; call Sendaheadb; call Sendbehindb; return;

Name: Forspll **Fig: 5,** 10
Admitted in: N NE SA CA
Input: -
Requires: Actcalled
If branch: Yes **Condition:** Busy?
Effect: 1. If busy, call Buslead4;
2. Else set busy; set forspl(1); set time in Norepforsl;
3. Endif; call Sendbehindl; return;

Name: Fs 1 set **Fig: 5,** 10, 11
Admitted in: N NE SA CA
Input: -
Requires: Sendbehinda and pos = 1
If branch: Yes **Condition:** Busy?
Effect: 1. If busy, call Buslead1 ;
2. Else set busy; set forspl(1); set time in Norepforsl;
3. Endif; return;

Name: Fs2set **Fig: 5,** 11
Admitted in: N NE SA CA
Input: -
Requires: Sendaheadbl and not busy, sendbehindbl
If branch: No **Condition: -**
Effect: Set busy; set forspl(2); set flag = 1; return;

Name: Fs3set **Fig: 5,** 10, 11
Admitted in: N NE SA CA
Input: -
Requires: Sendaheada or (Sendbehinda and pos > 1)
If branch: No **Condition: -**
Effect: Set busy; set forspl(3); set time in norepfors2; return;

Name: Forsplitdone **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Sepmeasfs1 or Waitsep
If branch: Yes **Condition:** Free agent?
Effect: 1. Send confirm-for-split;
2. If free agent, set flag = 4;
3. Else set flag = 5; set stime in Waitfast; reset flags in Splitagain and mergeagain;
4. Endif; return;

Name: Histfor **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Recfsplitover2
If branch: Yes **Condition:** More messages stored?
Effect: 1. If more messages stored, send forward oldest; delete it from store;
2. Else reset busy; reset flag in recfsplitover2;
3. endif; return;

Name: Lastbusy **Fig: 5, 8**
Admitted in: N NE SA CA
Input: forspl_b (pos-x)_r (x > 1) received
Requires: Messrec and pos = last
If branch: Yes **Condition:** Busy? and multiple.
Effect: 1. If busy, act on flag in buschek:
Merge(3): send Unmerge; store data; set forspl(6); set time in Recfsplitover2; reset flag in Recfsplitover2; break;
Split(3): store data; set forspl(6); set time in Recfsplitover2; reset flag in Recfsplitover2; break;
Forspl(6): Inform system; call stop mode;
2. Endifs; endif; return;

Name: Leadforsplit 1 **Fig: 5, 8**
Admitted in: N NE SA CA
Input: Message Forsplit-behind_ 1_f received by leader (pos = 1)
Requires: Messrec
If branch: Yes **Condition:** Busy?
Effect: 1. Setforspl(1); set time in Norepfors1 ;
2. If busy call Buslead2;
3. Else set busy;
4. Endif; return;

Name: Leadforsplit2 **Fig: 5, 8**
Admitted in: N NE SA CA
Input: Forsplit_behind_(N > 1)_f received by leader
Requires: Messrec
If branch: Yes **Condition:** Busy?
Effect: 1. Set forspl(1); set time in Norepforsl;
2. If busy call Buslead3;
3. Else set busy;
4. Endif; return;

Name: Mergeagain **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Splitagain
If branch: Yes **Condition:** Flag set? merge-hist received?
Effect: 1. If flag set, then if more stored history, read it;
2. If item received is confirm-merge,
3. Reset busy; reset forspl(2); reset flag; set target speed back to normal;
4. Endif; else set merge(2); reset flag; set target speed back to normal;
5. Endif; else if item received is merge-hist, set flag;
6. Endif; endif; return;

Name: Newlead **Fig: 5**
Admitted in: N NE SA **CA**
Input: -
Requires: Buschek, forspl(2), flag = 1
If branch: Yes **Condition:** Fault#l set?
Effect: 1. Update state vector; reduce target speed;
2. If fault #1 set, flag = 3; time to waitsep;
3. Else flag = 2;
4. Endif; return;

Name: Norepfors 1 **Fig: 5**
Admitted in: N NE **SA CA**
Input: -
Requires: Recconspll
If branch: Yes **Condition:** (time - stime) excessive?
Effect: 1. If (time - stime) excessive, reset busy, reset forspl(1); send for-split-over;
2. Endif; return;
Name: Norepfors2 **Fig: 5**
Admitted in: N NE SA **CA**
Input: -
Requires: Recsplitoverl

If branch: Yes **Condition:** (time - stime) excessive?
Effect: 1. If (time - stime) excessive, reset busy, reset forspl(3);
2. Endif; return;

Name: Norepfors3 **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Recsplitover2
If branch: Yes **Condition:** (time - stime) excessive?
Effect: 1. If (time - stime) excessive, reset forspl(4); set merge(l); set flag = 2; send forsp-over; reset flag in Stillgo;
2. Else call sepmeasfs2;
3. Endif; return;

Name: Norepfors4 **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Recsplitover3
If branch: Yes **Condition:** (time - stime) excessive?
Effect: 1. If (time - stime) excessive, reset forspl(4); set merge(l); set flag = 3; send confirm-merge; send forsp-over; set time in Norepm3; set target speed back to normal;
2. Endif; return;

Name: Norepfors3 **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Recsplitover2
If branch: Yes **Condition:** (time - stime) excessive?
Effect: 1. If (time - stime) excessive, reset forspl(5); set split(2); set flag = 3; send fsplitover; reset flag in Spliton;
2. Else call sepmeasfs3;
3. Endif; return;

Name: Norepfors6 , **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Recsplitover5
If branch: Yes **Condition:** (time - stime) excessive?
Effect: 1. If (time - stime) excessive, reset forspl(5); reset busy; send confirm-split; send forsp_over; set target speed back to normal;
2. Endif; return;

Name: Qfalt 1 **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Buschek and forspl(1) set
If branch: Yes **Condition:** Mode SA or CA?
Effect: 1. If mode is SA or CA, send Crashstopbehind;
2. Else call Recconfspl1 ;
3. Endif; return;

Name: Qmess **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Buschek and forspl(6)
If branch: Yes **Condition:** Message received?
Effect: 1. If message received, store it and add it as data to call to Recfsplitover2;
2. Else add Null mark to call;
3. Endif; call Recfsplitover2; return;

Name: Qslowmode **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Buschek and forspl(2), flag = 3 (therefore fault #1)
If branch: Yes **Condition:** Mode SA or CA?
Effect: 1. If mode = CA or SA, send Crashstopbehind;
2. Else call Waitsep;
3. Endif; return;

Name: Recaheadb **Fig: 5, 8**
Admitted in: N NE SA **CA**
Input: Message Forsplit_behind_(pos-1)-r received
Requires: Messrec
If branch: Yes **Condition:** Busy?
Effect: 1. Set forspl(2); set flag = 1;
2. If busy, call Sendunmerge2;
3. Else set busy;
4. Endif; reset message flag; return;

Name: Recbehinda **Fig: 5, 8**
Admitted in: N NE SA CA
Input: Veh (pos > 1) receives fspl_b_(pos)_f
Requires: Messrec
If branch: No **Condition: -**
Effect: Set busy; set forspl(3); set time in norepfors2; reset message flag; return;

Name: Recconfsll **Fig: 5**
Admitted in: N NE SA **CA**
Input: -
Requires: Qfaltl
If branch: Yes **Condition:** Confirm-forsplit received?
Effect: 1. If confirm-forsplit received, reset busy; reset message flag; reset forspl(1); send for-split-over;
2. Else call norepforsl;
3. Endif; return;

Name: Recconfspl2 **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Buschek and forspl(4)
If branch: Yes **Condition:** Confirm-for-split received?
Effect: 1. If confirm-forced-split received, reset forspl(4); set merge(l); set flag = 2; send fsplitover; reset flag in Stillgo; reset message flag;
2. Else call Norepfors3;
3. Endif; return;

Name: Recconfspl3 **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Buschek and forspl(4), flags = 2
If branch: Yes **Condition:** Confirm-for--split received?
Effect:
1. If confirm-for-split received, reset forspl(4); set merge(l) ; set flag = 3; send confirm-merge; send forsp-over; set time in Norepm3; set target speed back to normal;
2. Else call Norepfors4;
3. Endif; return;

Name: Recconfspl4 Fig: 5
Admitted in: N NE SA CA
Input: -
Requires: Buschek and forspl(5)
If branch: Yes **Condition:** Confirm for-split received?
Effect:
1. If confirm-forced-split received, reset forspl(5); set split(2); set flag = 3; send fsplitover; reset flag in Spliton; reset message flag;
2. Else call Norepfors5;
3. Endif; return;

Name: Recconfspl5 **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Buschek and forspl(5), flag = 2
If branch: Yes **Condition:** Confirm-for-split received?
Effect:
1. If confirm-for-split received, reset forspl(5); reset busy; send confirm-split; send forsp-over; set target speed back to normal;
2. Else call Norepfors6;
3. Endif; return;

Name: Recfsplitover1 **Fig: 5**
Admitted in: N NE SA **CA**
Input: -
Requires: Buschek, forspl(3)
If branch: Yes **Condition:** For-split-over received?
Effect:
1. If for-splitover received, reset busy, reset forspl(3); reset message flag;
2. Else call norepfors2;
3. Endif; return;

N a m e : Recfsplitover2 , **Fig: 5**
Admitted in: N NE SA CA
Input: Message received by Qmess or null
Requires: Qmess
If branch: Yes **Condition:** For-split-over received? Time elapsed? Flag set?

Effect: 1. If flag set, call Histfor;
2. Else if for-split-over received, set flag;
3. Else if (time - stime) excessive, set flag;
4. Endif; endif; return;

Name: Sendaheada **Fig: 5,** 10, 11
Admitted in: N NE SA CA
Input: -
Requires: Fault4or5 or Recackcont
If branch: Yes **Condition:** Pos = 1?
Effect: 1. Send backward message forsplit_behind_(pos)_r;
2. If pos = 1, call Fslset;
3. Else call Fs3set; send forward message forsplit_behind_ (pos)_f;
4. Endif; return;

Name: Sendaheadb **Fig: 5,** 10
Admitted in: N NE SA **CA**
Input: -
Requires: Forspla
If branch: No **Condition: -**
Effect: Send forward message forsplit_behind_(pos-1)_f; return;

Name: Sendbehindb **Fig: 5,** 10
Admitted in: N NE SA CA
Input: -
Requires: Forspla
If branch: No **Condition: -**
Effect: Send back message forsplit-behind-@os-1)-r; return;

Name: Sendbehindbl . **Fig: 5,** 11
Admitted in: N NE SA CA
Input: -
Requires: Nocontmess (11)
If branch: Yes **Condition:** Pos = last?
Effect: 1. Send forward message forsplit_behind_(pos-1)_f;
2. If pos = last, then if busy call Sendunmerge1;
3. Else call Fs2set;
4. Endif; call fs2set; send back message forsplit_behind_ (pos-1)_b;
5. Endif; return;

Name: Sendbehindl Fig: 5, 10
Admitted in: N NE SA CA
Input: -
Requires: Forspll (and so pos = 1)
If branch: No **Condition: -**
Effect: Send fspl_b_1 r back; return;

Name: Sepmeasfs1 **Fig: 5**
Admitted in: N NE SA CA
Input: Reading, d, of forward probe
Requires: Buschek and forfpl(2), flag = 2
If branch: Yes **Condition:** d = platoon spacing?
Effect: 1. If d > = platoon spacing call Forsplitdone;
2. Endif; return;

Name: Sepmeasfs2 **Fig: 5**
Admitted in: N NE SA
Input: Distance d from preceding vehicle - probe reading
Requires: Norepfors3
If branch: Yes **Condition:** d < 1.5*(in-platoon spacing)?
Effect: 1. If d < 1.5*(in-platoon spacing); set flag = 2; set time in Norepfors4 equal to that in Norepfors3;
2. Endif; return;

Name: Sepmeasfs3 **Fig: 5**
Admitted in: N NE SA
Input: Distance d from preceding vehicle - probe reading
Requires: Norepfors5
If branch: Yes **Condition:** d > = platoon spacing?
Effect: 1. If d > = platoon spacing; set flag = 2; set time in Norepfors6 equal to that in Norepfors5;
2. Endif; return;

Name: Splitagain1 **Fig:** 5
Admitted in: N NE SA CA
Input: -
Requires: Testactflag and activity flag set (therefore free agent)
If branch: Yes **Condition:** Act flag set? Split-hist received?

Effect: 1. If more stored history, read item;
2. If flag set, then if item is split-hist, reset flag;
3. Endif; else if item is confirm-split; reset busy; reset forspl(2); set target speed back to normal;
4. Endif; endif; else if flag set, ;
5. Else set split(l); reset flag; reset forspl(2); set target speed back to normal;
6. Endif; endif; return;

Name: Splitagain **Fig:** 5
Admitted in: N NE SA CA
Input: -
Requires: Buschek, forspl(2) flag = 5
If branch: Yes **Condition:** Flag set? Split-hist received?
Effect: 1. If more stored history, read item;
2. If flag set; then if item is confirm-split, reset busy; reset forspl(2); set target speed back to normal;
3. Endif; else if item is split-hist, set flag;
4. Endif; endif; else if flag set, set split(l); reset flag; reset forspl(2); set target speed back to normal;
5. Endif; endif; return;

Name: Sendunmerge1 **Fig:** 5, 11
Admitted in: N NE SA CA
Input: -
Requires: Sendaheadbl and busy (implies pos = last)
If branch: Yes **Condition:** Multiple
Effect: 1. Set forspl(2); set flag = 1;
2. Action according to flag in buschek:
Merge(3): reset merge(3); send unmerge; break;
Split(3): store data; set active flag; break;
Forspl(6): inform system; call stop mode;
3. Endifs; return;

Name: Sendunmerge2 **Fig:** 5
Admitted in: N NE SA CA
Input: -
Requires: Recaheadb and busy (implies pos = last)

If branch: Yes **Condition:** Multiple
Effect: 1. Action according to flag in buschek:

```
                Merge(3): reset merge(3); send Unmerge; break;
                Split(3): store data; set active flag; break;
                Forspl(6): inform system; call stop mode;
    2. Endifs; return;
```

Name: Testactflag **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Buschek and forspl(2), flag = 4 (therefore free agent)
If branch: Yes **Condition:** Active flag set?
Effect: 1. If active flag set, call splitagain1 ;

```
    2. Else reset busy; reset forsplit(2); target speed back to normal;
    3. Endif; return;
```

Name: Waitfast **Fig: 5**
Admitted in: N NE SA **CA**
Input: -
Requires: Mergeagain
If branch: Yes **Condition:** (time - stime) excessive?
Effect: (Note: Wait interval is for message to pass down platoon and back - measured in millisec.

```
    1. If (time - stime excessive) reset busy; reset forspl(2); target speed back to normal;
    2. Endif; return;
```

Name: Waitsep **Fig: 5**
Admitted in: N NE SA CA
Input: -
Requires: Qslowmode
If branch: Yes **Condition:** (time - stime) excessive?
Effect: 1. If (time - stime) excessive, call Forsplitdone;

```
    2. Endif; return
```

AS.5 EMER-CHANGE (Figure A.6)

Name: Busychek **Fig:** 1,2,3,4,5,6
Specification in: Sup (1)

Name: Callstop **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Passemch
If branch: Yes **Condition:** Flag? gate near? **SA speed?**
Effect: 1. If flag set, then if gate near ;
2. Else if maxspeed is **SA** speed, call Stop mode; reset any lanes put into **SA;**
3. Else call Sacall for each lane affected;
4. Endif; send unconfirm_emer_change; message emerch_fail to system; reset busy;
5. Else if gate near set flag;
6. Endif; endif; return;

Name: Compposn **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Buschek, emerch(1) and flag = 7
If branch: Yes **Condition:** result of calculation
Effect: From known speed and positions of other vehicles predict time at a loc some way ahead at which all partners should be in position; send this target to partners via messages emerch_target; reset flag0 in Qallrec; if position sent is before next gate set flag1 in Qallrec = 0; else set flag1 = 1; endif; reset flag in neset3; Flag = 8; return;

Name: Emerchcall **Fig: 6,** 10
Admitted in: N NE SA CA
Input: -
Requires: Actcalled
If branch:? Yes **Condition:** Fault #6 or #7?
Effect: 1. Set busy; set emerch(1); set flag = 1; call Nesetl;
2. If fault #6 or fault #7 is present, reroute reception and transmission channels so that side transmission becomes system transmission and side reception system reception, set flag in Sendreqemerch;
3. Else reset flag in Sendreqemerch;
4. Endif; return;

Name: Forsplitcall **Fig: 6**
Admitted in: N NE SA CA
Input: -
Requires: Inposn
If branch: Yes **Condition:** For-split called?
Effect:
1. If for-split called (ie forspl(1) in buschek set), send for-split-called to changer; reset emerch(x);
2. Else call recforsplcal3;
3. Endif; return;

Name: Inposn **Fig: 6**
Admitted in: N NE SA CA
Input: -
Requires: Buschek, emerch(x) (2 <=x c =5) and flag = 7
If branch: Yes **Condition:** In target position? Gate?
Effect: 1. Call forsplitcall1;
2. If loc = target position, then
3. If gate near, send message (message depends on x, see below); flag = 8; reset flag in Norepem3;
4. Endif; endif; return;

Value of x	Message
2	adj_ahead_ok
3	adj_behind_ok
4	next_ahead-ok
5	next_behind_ok

Name: Neset 1 **Fig: 5, 10**
Admitted in: N NE SA CA
Input: -
Requires: Emerchcall
If branch:? Yes **Condition:** Mode = CA? Fault = #6, #7?
Effect: 1. If fault #6 or #7 set, then if mode = CA, ;
2. Else mode = NE from previous gate to one two ahead; maxspeed unchanged;
3. Endif; endif; call Sendreqemerch; return;

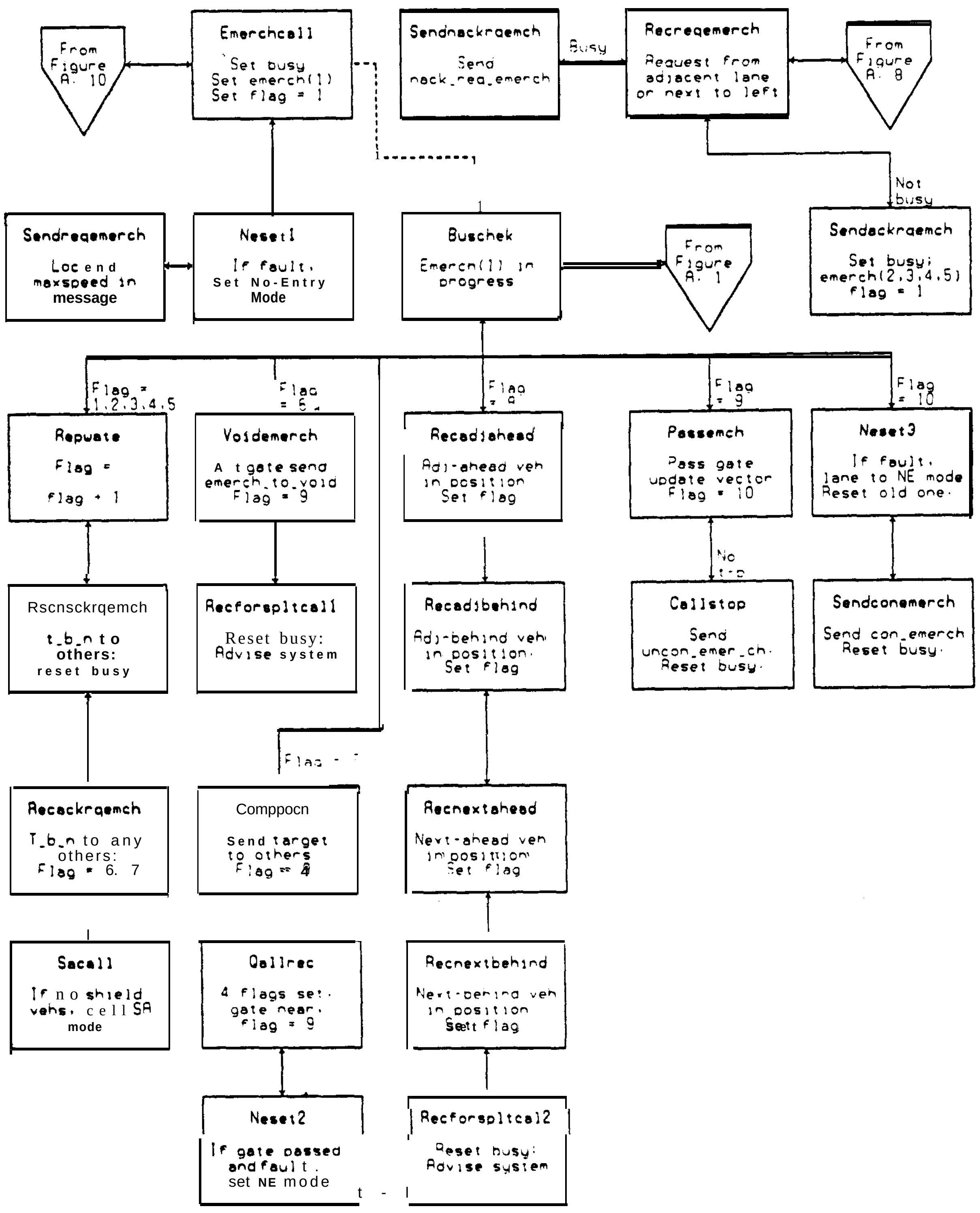


Figure A.6. Emergency Change (initiator).

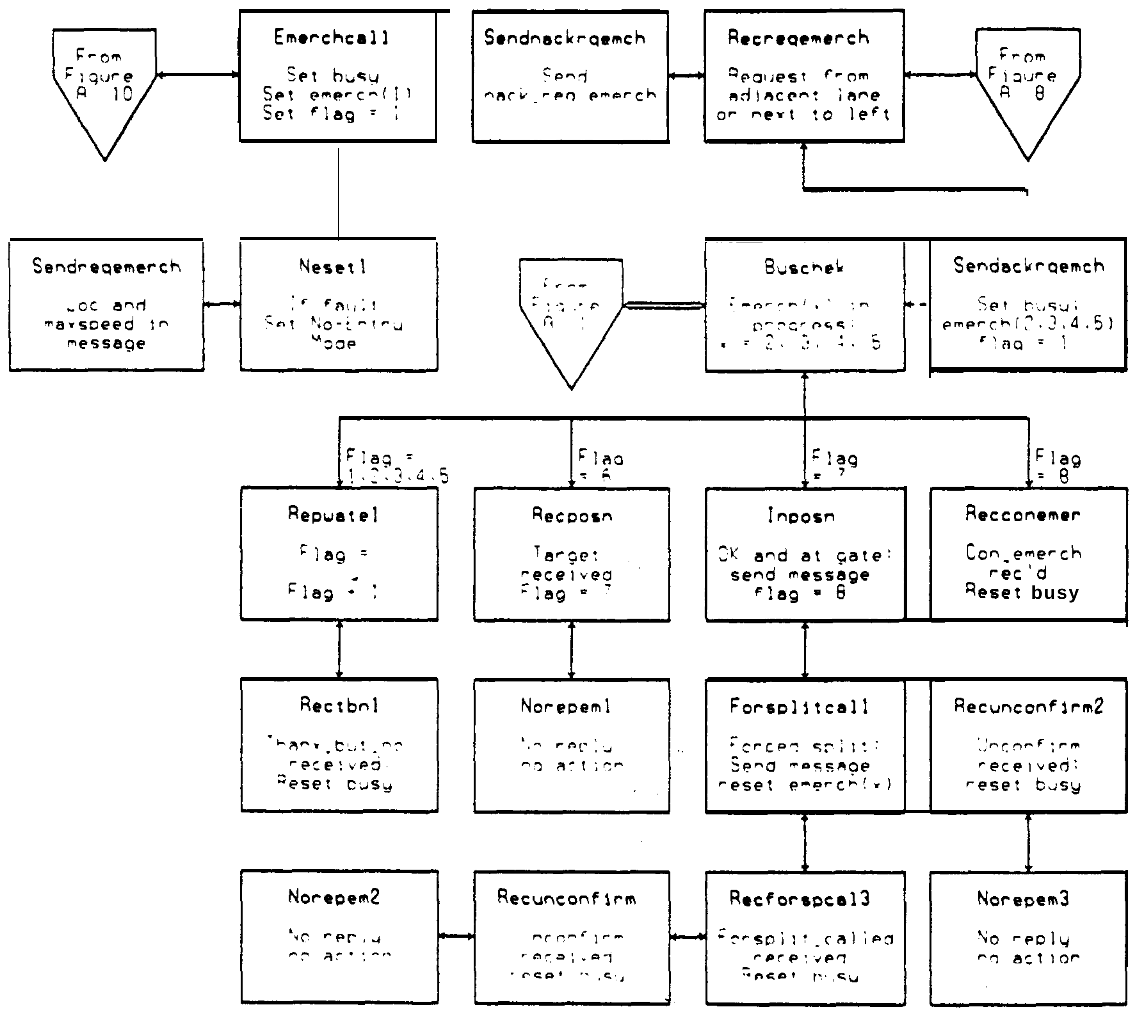


Figure A. 6a. Emergency Change (participant).

Name: Neset2 **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Recforsplitcal2
If branch: Yes **Condition:** Faults? Flag?
Effect: 1. If fault #6 or #7, then if flag set, then if gate near, ;
2. Else set NE (and if necessary SA in parallel lanes) for two gates ahead; restore previous mode for section behind gate just passed;
3. Endif; else if gate near, set flag;
4. Endif; endif; endif; call Qallrec; return;

Name: Neset3 **Fig: 6**
Admitted in: N SA
Input: -
Requires: Buschek. emerch(1) and flag = 10
If branch: Yes **Condition:** Faults?
Effect: 1. If fault #6 or #7, set new lane to NE mode, no change in maxspeed; reset lane to left to original mode and original speed; reset any lane set to SA to previous value;
2. Endif; call Sendconemerch; return;

Name: Norepm 1 **Fig: 6**
Admitted in: N NE SA CA
Input: -
Requires: Recposn
If branch: Yes **Condition:** Flag? time excessive;
Effect: 1. If flag set, then if (time - stime) excessive, reset emerch(x); reset busy;
2. Endif; else set flag; set stime = time;
3. Endif; return;

Name: Norepm2 **Fig: 6**
Admitted in: N NE SA **CA**
Input: -
Requires: Recunconfirm
If branch: Yes **Condition:** Flag? time excessive;
Effect: 1. If flag set, then if (time - stime) excessive, reset emerch(x); reset busy;
2. Endif; else set flag; set stime = time;
3. Endif; return;

Name: Norepm3 **Fig: 6**
Admitted in: N NE SA CA
Input: -
Requires: Recunconfirm2
If branch: Yes **Condition:** Flag? time excessive;
Effect: 1. If flag set, then if (time - stime) excessive, reset emerch(x); reset busy;
2. Endif; else set flag; set stime = time;
3. Endif; return;

Name: Passemerch **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Buschek, emerch(1) flag = 9
If branch: Yes **Condition:** Turning point here?
Effect: 1. If turning point here change lane; update vector from gate data; set flag= 10;
2. If new ln# = 0, send fault-out to system;
3. Endif; else call Callstop;
4. Endif; return;

Name: Qallrec **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Neset2
If branch: Yes **Condition: 4** flags? flag0? flagl? gate near?
Effect: 1. If flag0 is set, then if flag1 = 0, then
2. If gate is near, then if all 4 flags are set,
3. Send emerch_at_gate_x; set counter in Passemch; Flag = 9;
4. Endif; else send unconfirm_emerch; reset busy; reset emerch(1);
5. Endif; else if gate is near ;
6. Else set flag1 = 0; reset flag0;
7. Endif; else if gate is near set flag0;
8. Endif; endif; return;

Name: Recadjahead **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Buschek, emerch(1) and flag = 8
If branch: Yes **Condition:** Adj_ahead_ok received?
Effect: 1. If adj_ahead_ok received, reset message flag; set flag in Qallrec;
2. Endif; call recadjbehind; return;

Name: Recadjbehind **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Recadjahead
If branch: Yes **Condition:** Adj_behind_ok received?
Effect: 1. If adj_behind_ok received, reset message flag; set flag in Qallrec;
2. Endif; call recnextahead; return;

Name: Recackrqemerch **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Recnackrqemch
If branch: Yes **Condition:** Replies received each position?
Effect: 1. If no ack_req_emerch messages received, set flag = 6; if speed of vehicle is low, call Sacall for adj and next;
2. Endif; else for each position (adj or next, ahead or behind):
a. If no relevant acks, set flag in Qallrec; if this is a behind position and speed of vehicle is low, call Sacall;
b. Endif; endif; send thanx_but_no to all irrelevant respondents;
3. Set flag = 7; endif; return;

Name: Recconemer **Fig: 6**
Admitted in: N NE SA CA
Input: -
Requires: Buschek emerch(x) (2<=x<=5) and flag = 8
If branch: Yes **Condition:** Con-emerch received?
Effect: 1. If con-emerch received, then reset emerch(x); reset busy; revert to normal speed;
2. Else call Recunconfirm2;
3. Endif; return;

Name: Recposn **Fig: 6**
Admitted in: N NE SA CA
Input: Speed, posn, ln# of caller
Requires: Buschek, emerch(x), (2 <=x<=5) and flag = 6
If branch: Yes **Condition:** emerch-target received?
Effect: 1. If emerch-target received, store target and caller's details; reset flag in Norepem2; flag = 7;
2. Else call Norepm 1;
3. Endif; return;

Name: Rectnb 1 **Fig: 6**
Admitted in: N NE SA CA
Input: -
Requires: Repwatel , flag = 5
If branch: Yes **Condition:** Thanx_but_no received?
Effect: 1. If thanx_but_no received, reset emerch(x), reset busy;
2. Endif; return;

Name: Repwate **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Buschek, emerch(1) and flag = 1,2,3,4,or 5.
If branch: Yes **Condition:** Flag = 5?
Effect: 1. If flag = 5, call recnackrqemch;
2. Endif; flag = flag + 1; return;

Name: Repwatel **Fig: 6**
Admitted in: N SA
Input: -
Requires: Buschek, emerch(x) (2 <=x<=5) and flag = y (1 < = y <=5)
If branch: Yes **Condition:** Flag = 5?
Effect: 1. If flag = 5, call Rectbnl; reset flag in Norepem 1;
2. Endif; flag + = 1; return;

Name: Recforsplitcal1 **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Voidemerch
If branch: Yes **Condition:** For-split called?
Effect: 1. If for-split-called is received, reset message flag; send for-split-called to other partners in emerch; reset emerch(1); advise system; reset busy;
2. Endif; return;

Name: Recforsplitcal2 **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Recnextbehind
If branch: Yes **Condition:** For-split called?
Effect: 1. If for_split_called is received, reset message flag; send for-split-called to other partners in emerch; reset emerch(1); reset busy; advise system;
2. Endif; call neset2; return;

Name: Recforsplitcal3 **Fig: 6**
Admitted in: N NE SA CA
Input: -
Requires: Forsplitcall
If branch: Yes **Condition:** For-split called?
Effect: 1. If for_split_called is received, reset message flag;
reset busy; reset emerch(x);
2. Endif; call recunconfirm; return;

Name: Recnextahead **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Recadjbehind
If branch: Yes **Condition:** Next-ahead-ok received?
Effect: 1. If next-ahead-ok received, reset message flag; set flag in Qallrec;
2. Endif; call recnextbehind; return;

Name: Recnextbehind **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Recnextahead
If branch: Yes **Condition:** Next-behind-ok received?
Effect: 1. If next-behind-ok received, reset message flag; set flag in Qallrec;
2. Endif; call recforsplitcal2; return;

Name: Recnackreqemch **Fig: 5**
Admitted in: NE CA
Input: -
Requires: Repwate, flag = 5
If branch: Yes **Condition:** Relevant nack_rq_emerch
received?
Effect:
1. If a relevant nack_req_emerch received, (relevant means that there is not another platoon behind the vehicle but ahead of the nacker), then
2. If it is a nack because it system mode makes it impossible to reach this speed, reset own target speed;
3. Endif; send thanx_but_no to all others who have sent messages; reset busy; reset emerch(1); reset all message flags;
4. Else call Recackreqemch;
5. Endif; return;

Name: Recreqemerchange **Fig: 6, 8**
Admitted in: N NE SA CA
Input: Loc, ln# and flag from sender
Requires: Messrec and request-emerchange received
If branch:? Yes **Condition: -** Busy? NE or CA?
Effect: I. Reset message flag;
2. If less than platoon spacing ahead of loc of sender or if less than two spacings behind in lane to immediate right or next, then
3. If busy or if fault set or if mode = CA or NE or if mode = **SA** and stated speed of sender too high, call Sendnackrqemerch;
4. Else call Sendackrqemerch; if fault #6 or #7 is set,
5. Reroute transmission to system transmission and so for reception; set flag;
6. Endif; endif; endif; return;

Name: Recunconfirm **Fig: 6**
Admitted in: N NE SA CA
Input: -
Requires: Recforsplcal3
If branch: Yes **Condition:** Unconfirm_emerch received?
Effect: 1. If unconfirm-emerch received, reset busy; reset emerch(x);
2. Else call Norepem2;
3. Endif; return;

Name: Recuncontirm2 **Fig: 6**
Admitted in: N NE SA CA
Input: -
Requires: Reccomemer
If branch: Yes **Condition:** Unconfirm_emerch received?
Effect: 1. If unconfirm-emerch received, reset busy; reset emerch(x);
2. Else call Norepem3;
3. Endif; return;

Name: Sacall **Fig: 6,** Gen
Admitted in: NE CA
Input: Lane affected and loc
Requires: Recackrqemch
If branch: No **Condition: -**
Effect: From gate behind stated position to two ahead, lane stated to SA mode; return;

Name: Sendackrqemerch **Fig: 6, 8**
Admitted in: N SA
Input: -
Requires: Recreqemerch
If branch:? No **Condition: -**
Effect: 1. Set busy;
set emerch(2) (forward and in adjacent lane),
emerch(3) (rearward and in adjacent lane),
emerch(4) (forward in next lane), or
emerch(5)(rearward in next lane);
2. Set flag = 1;
3. SendAck_request_emerchange with loc and ln# after x-lchrons for emerch(x);
return;

Name: Sendconemerch **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Neset3
If branch: No **Condition: -**
Effect: Send confirm-emerch; reset busy; reset emerch(1); reset any lanes put into SA;
return;

Name: Sendnackrqemerch **Fig: 6, 8**
Admitted in: N NE SA CA
Input: -
Requires: Recreqemerch
If branch: No **Condition: -**
Effect: Send nack_request_emer_chng with loc, lane# and maxspeed; return;

Name: Sendreqemerch **Fig: 5,** 10
Admitted in: NE CA
Input: -
Requires: Neset1
If branch:? No **Condition: -**
Effect: Send Request-emerchange with lane, loc and maxspeed; return;

Name: Voidemerch **Fig: 6**
Admitted in: NE CA
Input: -
Requires: Buschek, emerch(6), flag = 6
If branch: Yes **Condition:** Gate near?
Effect: 1. If gate near, send emerch_to_void naming gate; flag = 9; reset flag in Callstop;
2. Else call Recforsplitcalll;
3. Endif; return;

A.6 PLATOON-LEADER'S PROBE (Figure A.7)

Name: Forprob Fig: 1, 7
Specification in: SUP

Name: Insight **Fig: 7**
Admitted in: N NE SA CA
Input: Distances of vehicles seen in probe (or null)
Requires: Forprob and flag = 1
If branch: Yes **Condition:** Vehicles visible?
Effect: 1. If any vehicle can be seen, select one most likely to be in same lane;
2. Send probe-l, including state vector and range of vehicle perceived (using corrected loc stored in Startprobe); set flag = 2;
3. Else send probe-nov, giving state vector; set flag = 9;
4. Endif; return;

Name: Nofault **Fig: 7**
Admitted in: N NE SA CA
Input: Message No-fault-l from system
Requires: Forprob, flag = 16, 17, 18
If branch: Yes **Condition:** Message No-fault-l rec'd?
Effect: 1. If message No-fault-l received, reset message flag; reset flag;
2. Else flag += 1;
3. Endif; return;

Name: Novehnorep **Fig: 7**
Admitted in: N NE SA CA
Input: -
Requires: Forprob, flag = 12
If branch: No **Condition: -**
Effect: Reset flag in forprob; return;

Name: Probeagain **Fig: 7**
Admitted in: N NE SA CA
Input: -
Requires: Forprob, flag = 5
If branch: No **Condition: -**
Effect: Send probe-;!, giving state vector; flag = 6; return;

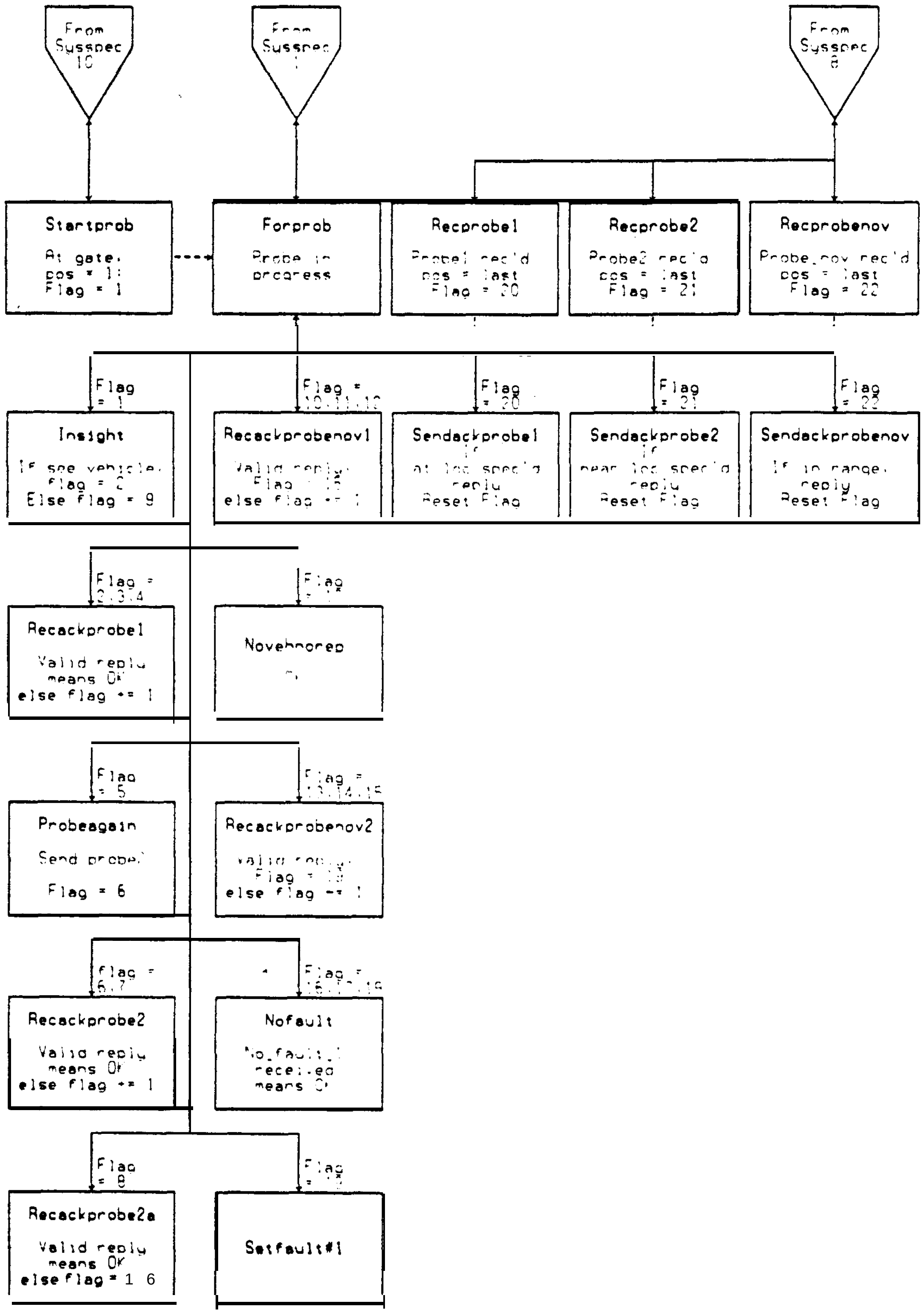

From Sysspec 10
From Sysspec 1
From Sysspec 8
Startprob
At gate, pos = 1; Flag = 1
Forprob
Probe in progress
Recprobe1
Probe1 rec'd pos = last Flag = 20
Recprobe2
Probe2 rec'd pos = last Flag = 21
Recprobenov
Probe_nov rec'd pos = last Flag = 22
Flag = 1
Flag = 10,11,12
Flag = 20
Flag = 21
Flag = 22
Insight
If see vehicle, flag = 2 Else flag = 9
Recackprobenov1
Valid reply, Flag = 13 else flag += 1
Sendackprobe1
If at loc spec'd reply Reset Flag
Sendackprobe2
If near loc spec'd reply Reset Flag
Sendackprobenov
If in range, reply Reset Flag
Flag = 2,3,4
Flag = 1?
Recackprobe1
Valid reply means OK else flag += 1
Novehnoreo
Flag = 5
Flag = 13,14,15
Probeagain
Send probe1 Flag = 6
Recackprobenov2
Valid reply, Flag = 18 else flag += 1
Flag = 6,7
Flag = 16,17,18
Recackprobe2
Valid reply means OK else flag += 1
Nofault
No_fault_1 received means OK
Flag = 8
Flag = 19
Recackprobe2a
Valid reply means OK else flag = 1 6
Setfault#1

Figure A.7. Forward Probe.

Name: Recackprobenov 1 **Fig: 7**
Admitted in: N NE SA CA
Input: Location of sender
Requires: Forprob and flag = 9, 10, 11
If branch: Yes **Condition:** Ack_probe_nov received?
Effect: 1. If ack_probe_nov received, work out distance and likely angle from stated location;
2. If reply valid, flag = 15;
3. Endif; endif; reset message flag; flag + = 1; return;

Name: Recackprobenov2 **Fig: 7**
Admitted in: N NE SA CA
Input: Location of sender
Requires: Forprob and flag = 13, 14, 15
If branch: Yes **Condition:** Ack_probe_nov received?
Effect: 1. If ack_probe_nov received, work out distance and likely angle from stated location;
2. If reply valid, flag = 20;
3. Endif; endif; reset message flag; flag - = 1; return;

Name: Recackprobe1 **Fig: 7**
Admitted in: N NE SA CA
Input: Location of sender
Requires: Forprob, and flag = 2,3,4
If branch: Yes **Condition:** Ack_probe_1 received?
Effect: 1. If ack_probe_1 received, work out distance and likely angle from stated location;
2. If reply valid, reset flag;
3. Endif; endif; reset message flag; flag + = 1; return;

Name: Recackprobe2 **Fig: 7**
Admitted in: N NE SA CA
Input: Location of sender
Requires: Forprob, and flag = 6,7
If branch: Yes **Condition:** Ack_probe_2 received?
Effect: 1. If ackgrobe 2_received, work out distance and likely angle from stated location;
2. If reply valid (ie within 30 m or so), set flag = - 1;
3. Endif; endif; reset message flag; flag + = 1; return;

Name: Recackprobe2a **Fig: 7**
Admitted in: N NE SA CA
Input: Location of sender
Requires: Forprob, and flag = 8
If branch: Yes **Condition:** Ack_probe_2 received?
Effect: 1. If ack_probe_2 received, work out distance and likely angle from stated location;
2. If reply valid (ie within 30 m or so), reset message flag; reset flag;
3. Else if only one vehicle visible, send Fault1_1_veh to system; reset message flag; flag = 16;
4. Else flag = 19;
5. Endif; endif; else if only one vehicle visible, send Fault1_1 yeh to system; flag= 16;
6. Else flag = 19;
7. Endif; endif; return;

Name: Recprobe 1 **Fig: 7, 8**
Admitted in: N NE SA CA
Input: -
Requires: Messrec
If branch: Yes **Condition:** Probe-l received?
Effect: 1. If probe-l received,
2. Set flag in Forprob = 20;
3. Else reset message flag;
4. Endif; return;

Name: Recprobe2 **Fig: 7, 8**
Admitted in: N NE SA CA
Input: -
Requires: Messrec
If branch: Yes **Condition:** Probe-2 received?
Effect: 1. If probe-2 received
2. Set flag in Forprob = 2 1;
3. Else reset message flag;
4. Endif; return;

Name: Recprobenov **Fig: 7, 8**
Admitted in: N NE SA CA
Input: -
Requires: Messrec
If branch: Yes **Condition:** Probe-nov received?
Effect: 1. If probe-nov received,
2. Set flag in Forprob = 22;
3. Else reset message flag;
4. Endif; return;

Name: Sendackprobenov **Fig: 7**
Admitted in: N NE SA CA
Input: Loc of sender, stated range
Requires: Forprob, flag = 22
If branch: Yes **Condition:** Within 1.5*platoon spacing of sender?
Effect: 1. If within range (1.5 times platoon spacing, say) of sender,
2. Send Ack_probe_nov;
3. Endif; reset flag; reset message flag; return;

Name: Sendackprobe 1 **Fig: 7**
Admitted in: N NE SA CA
Input: Loc of sender, stated range
Requires: Forprob, flag = 20
If branch: Yes **Condition:** At location specified
Effect: 1. If tolerably near (10 metres say) to specified range from sender,
2. Send Ack_probe_1;
3. Endif; reset flag; reset message flag; return;

Name: Sendackprobe2 **Fig: 7**
Admitted in: N NE SA CA
Input: Loc of sender, stated range
Requires: Forprob, flag = 21
If branch: Yes **Condition:** At location specified
Effect: 1.If approximately (25 metres say) at specified range from sender,
2. Send Ack_probe_2;
3. Endif; reset flag; reset message flag; return;

Name: Setfault1 **Fig: 7**
Admitted in: N NE SA CA
Input: -
Requires: Forprob, flag = 19
If branch: No **Condition: -**
Effect: Set fault #1; reset flag; return;

Name: Startprobe **Fig: 7,** 10
Admitted in: N NE SA CA
Input: Loc of gate (gate is near)
Requires: Mersplit (10)
If branch: Yes **Condition:** Pos? Busy? Fault #1 **or** #3 **set?**
Effect: 1. If busy ;
2. Else if fault #1 or #3 set ;
3. Else if pos = 1, then store any correction to loc as revealed by gate reading; set flag in Forprob = 1;
4. Endif; endif; endif; return;

A.7 RECEIPT OF MESSAGES (Figure A.8)

Name: Buslead2 **Fig:** 5, 8
Specification in: Forced-Split (5)

Name: Buslead3 **Fig:** 5
Specification in: For-split (5)

Name: Cacall **Fig:** 8
Specification in: Gen

Name: Crashstopcall **Fig:** 8
Specification in: Gen

Name: Cutspeed **Fig:** 8
Specification in: Gen

Name: Lastbusy **Fig:** 5, 8
Specification in: For-split (5)

Name: Leadforsplit 1 **Fig:** 5
Specification in: Forced-split (5)

Name: Leadforsplit2 **Fig:** 5, 8
Specification in: Forced-split (5)

Name: Messrec **Fig:** 1,8
Specification in: SUP

Name: Necall **Fig:** 8
Specification in: Gen

Name: Norcall **Fig:** 8
Specification in: Gen

Name: Passon **Fig:** 8
Admitted in: N NE SA CA
Input: Message Request-merge rec'd, pos = last > 1
Requires: Messrec
If branch: Yes **Condition:** Same lane as requester?
Effect: 1.If in same lane as requester, pass message up platoon;
2. Endif; return;

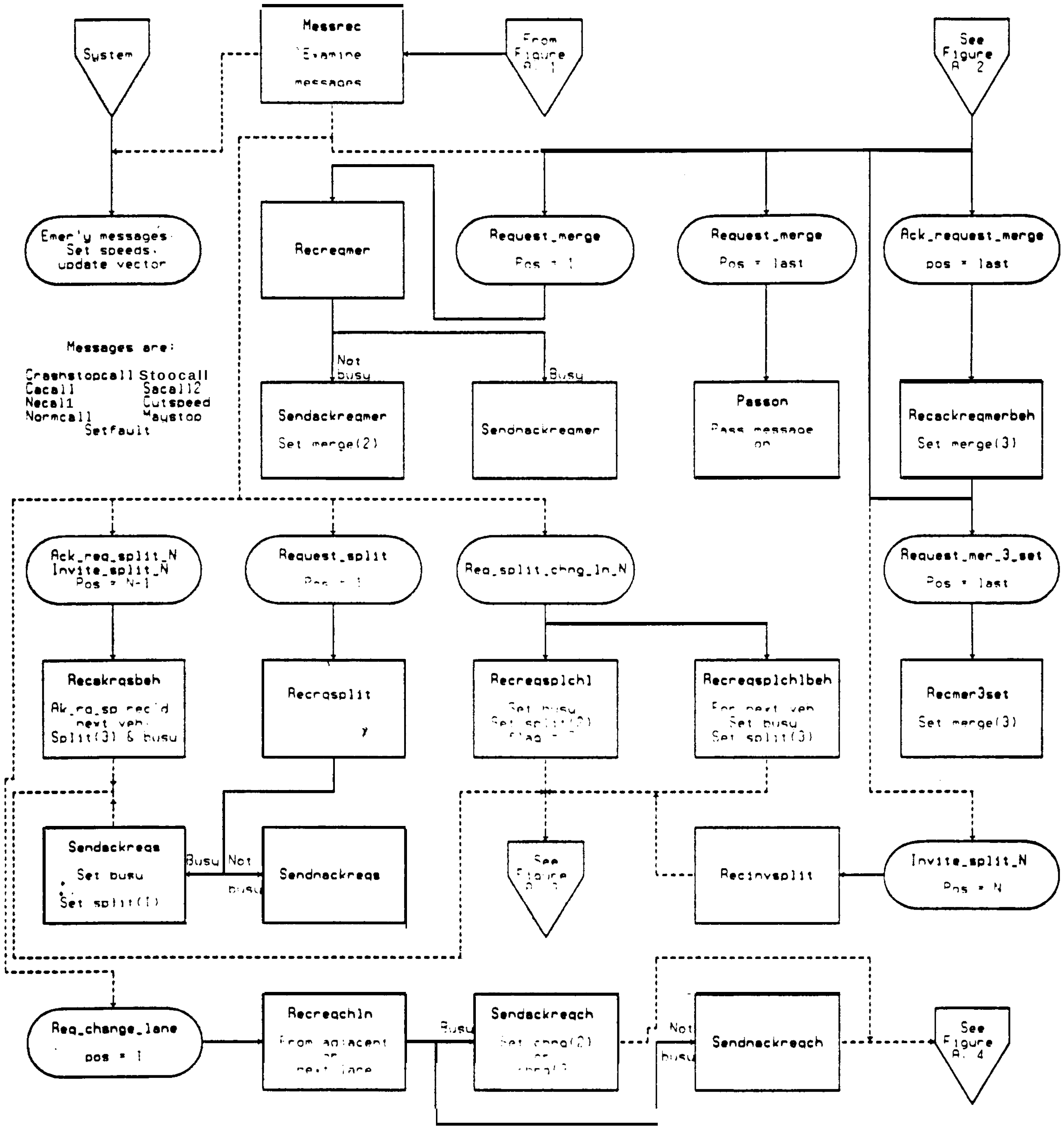


Figure A. 8. Messages.
(Messages responded to are in ovals.)

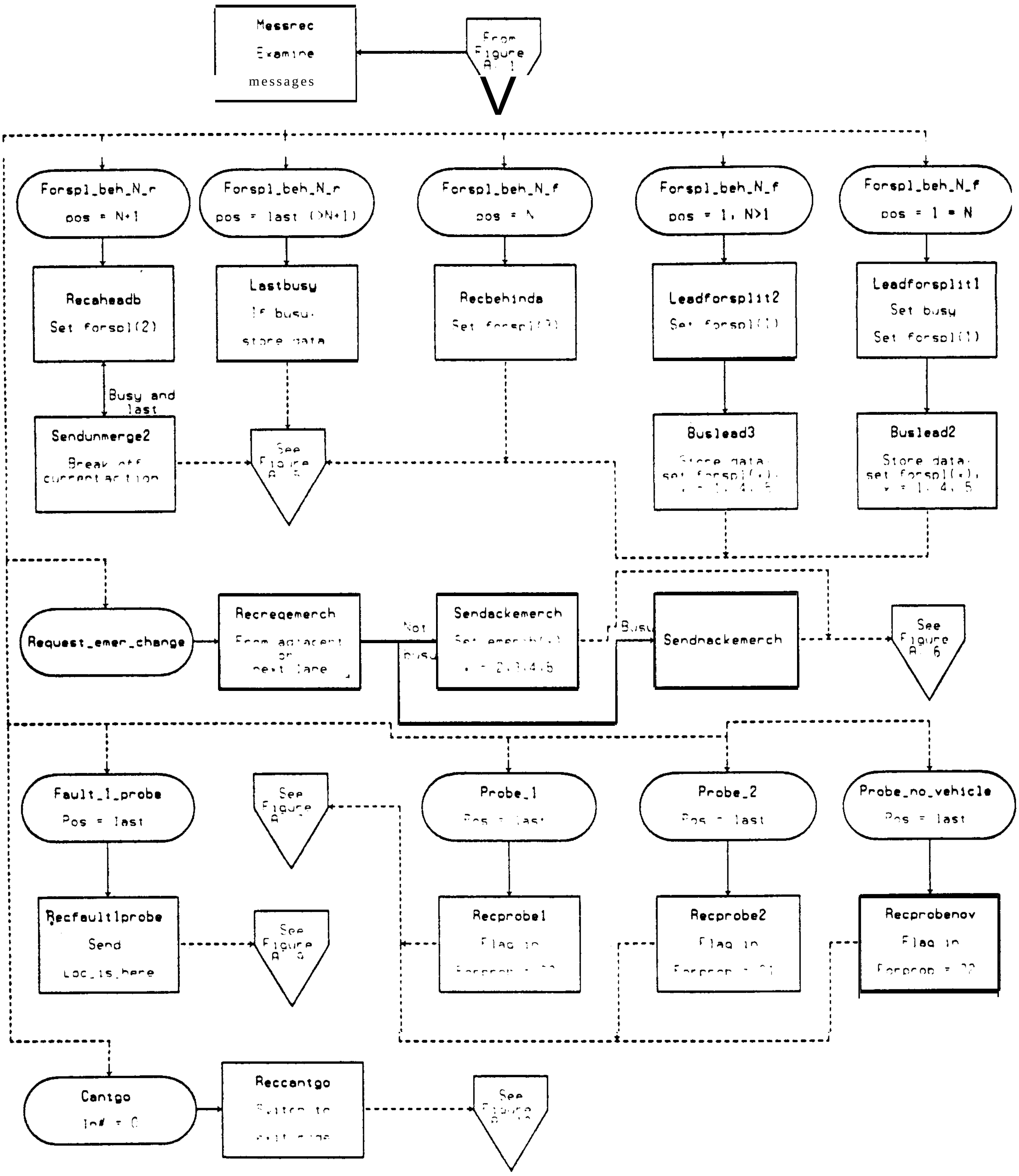


Figure A.8a. Messages (continued).
(Messages responded to are in ovals.)

Name: Recaheadb **Fig: 5, 8**
Specification in: For-split‘ (5)

Name: Recackreqmerbeh **Fig: 2, 16**
Specification in: Merge(2)

Name: Recakrqsbeh **Fig: 3, 8**
Specification in: Split (3)

Name: Recbehinda **Fig: 5, 8**
Specification in: For-split (5)

Name: Reccantgo **Fig: 8, 12**
Specification in: Joinquit (12)

Name: Recfault1probe **Fig: 8, 9**
Specification in: Faults (9)

Name: Recinv_spl **Fig: 3, 8**
Specification in: Split (3)

Name: Recprobel **Fig: 7, 8**
Specification in: Forqrob (7)

Name: Recprobe2 **Fig: 7, 8**
Specification in: Forprob (7)

Name: Recprobenov **Fig: 7, 8**
Specification in: Forqrob (7)

Name: Recreqsplit **Fig: 3, 8**
Specification in: Split (3)

Name: Recreqmer3set **Fig: 2, 8**
Specification in: Merge (2)

Name: Recreqchngln **Fig:** 4, 8
Specification in: Change-lane (4)

Name: Recreqemerchange **Fig: 6, 8**
Specification in: Emer-change (6)

Name: Recreq_mer **Specification in:** Merge (2)	**Fig:** 2, 8
Name: Recreqsplchl **Specification in:** Split (3)	**Fig:** 3, 8
Name: Recreqsplchlbeh **Specification in:** Split (3)	**Fig:** 3, 8
Name: Sacall2 **Specification in:** Gen	**Fig:** 8
Name: Sendackreqchln **Specification in:** Change-lane (4)	**Fig:** 4, 8
Name: Sendackrqemerch **Specification in:** Emer-change (6)	**Fig:** 6, 8
Name: Sendackreqmer **Specification in:** Merge (2)	**Fig:** 2, 8
Name: Sendackreqs **Specification in:** Split (3)	**Fig:** 3, 8
Name: Sendonf **Specification in:** For-split (5)	**Fig:** 5, 8
Name: Sendnackreqchln **Specification in:** Change-lane (4)	**Fig:** 4, 8
Name: Sendnackreqmer **Specification in:** Merge (2)	**Fig:** 2, 8
Name: Sendnackreqs **Specification in:** Split (3)	**Fig:** 3, 8
Name: Sendnackrqemerch **Specification in:** Emer-change (6)	**Fig:** 6, 8
Name: S topcall **Specification in:** Gen	**Fig:** 8

Name: (Intfalt) **Fig: 9**
Admitted in: N NE SA CA
Input: MON routines
Requires: Fault detected by internal monitor
If branch: Yes **Condition:** Fault #1? #11?
Effect: 1. Sets fault flag #1 - #11 as appropriate;
2. If fault #1, reset counter in Qtwofaults;
3. Endif; else if fault 11, and speed < c target, and Maystop flag reset, set Mystp flag;
4. Endif;

Note This routine is strictly part of MON

A.8 NEW FAULTS (Figure A.9)

Name: Newfault Fig: 1, 9
Specification in: Sup

Name: Nex ttum Fig: 9
Admitted in: N NE SA CA
Input: -
Requires: Reclocishere, no message received
If branch: No **Condition: -**
Effect: Decrement counter in Qtwofaults; return;

Name: Qtwofaults **Fig: 9**
Admitted in: N NE SA CA
Input: Fault flags
Requires: Newfalt, and some faults present
If branch: Yes **Condition:** Proscribed or 3+ faults?
Effect:
```
1. If more than two faults (etc) present, call Sendstopit;
    2. Endif; if one of pairs of faults, modes and flags listed below present, call
    Sendstopit;
    3. Endif; if fault #1 is present, then if counter is set,
    4. Call Reclocishere;
    5. Else call Sendfaultlprobe;
    6. Endif; return;
```

Note
The following pairs of faults, etc. can induce a hazard, and therefore should induce Stopit.

#1 with	#2, #3, or #10
#6 with	#8
#7 with	#9
#10 and	CA mode
Maystop flag and	#1 or #3

Name: Recfaultlprobe . **Fig: 8, 9**
Admitted in: N NE SA CA
Input: Message fault-lqrobe received, with loc
Requires: Messrec
If branch: Yes **Condition:** Same lane? In range? Pos?
Effect:
```
1. If pos = last, then
    2. If ln# and loc indicate that vehicle is in same lane as sender and not more than
    two platoon spacings ahead, send loc is_here (includes loc);
    3. Endif; endif; return;
```

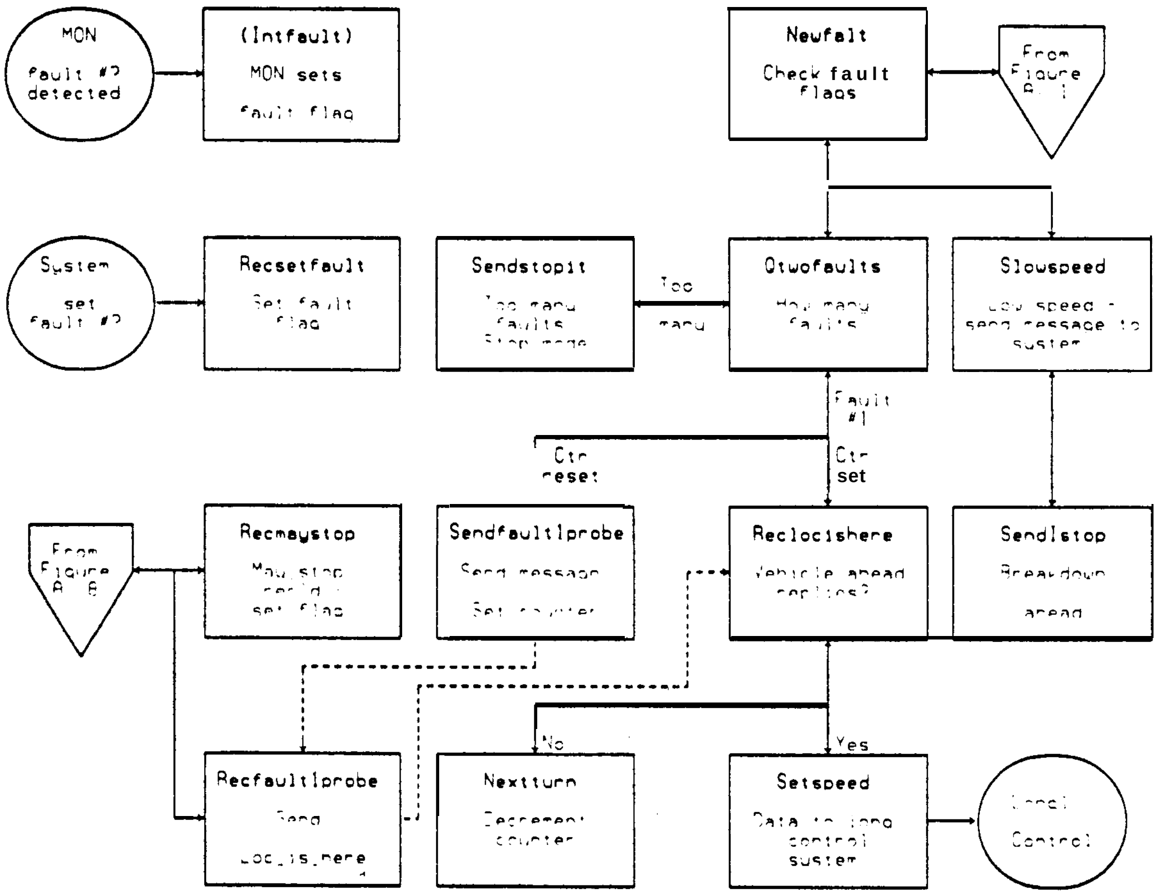


Figure A.9. Fault Flags.

Name: Reclocishere **Fig: 9**
Admitted in: N NE SA CA
Input: -
Requires: Qtwofaults, fault #1 and counter set
If branch: Yes **Condition:** Loc_is_here received?
Effect: 1. If loc_is_here received, call Setspeed, passing loc of sender;
2. Else call nextturn;
3. Endif; return;

Name: Recmaystop **Fig: 9**
Admitted in: N NE SA CA
Input: -
Requires: Message Maystop from system
If branch: No **Condition: -**
Effect: Sets relevant flag in Newfalt; return;

Note Maystop is sent when there is evidence of congestion.

Name: Recsetfault **Fig: 9**
Admitted in: N NE SA CA
Input: Message Setfault from system
Requires: Fault detected by system
If branch: Yes **Condition:** Fault #1, #11?
Effect: 1. Sets fault flag #1 - #1 1 as appropriate;
2. If fault #1, reset counter in Qtwofaults;
3. Endif; else if fault #11, and speed << target, and Maystop flag not set, set Maystop flag;
4. Endif; return;

Name: Setspeed **Fig: 9**
Admitted in: N NE SA CFA
Input: Loc and speed of sender of loc_is_here
Requires: Reclocishere
If branch: No **Condition: -**
Effect: Set maxspeed to maintain 1.5*(normal platoon spacing), using normal longitudinal control algorithm;

Name: Sendfaultlprobe **Fig: 9**
Admitted in: N NE SA CA
Input: -
Requires: Qtwofaults, fault #1 present, counter reset
If branch: No **Condition: -**
Effect: Send fault-lqrobe with loc and ln#; set counter in Qtwofaults = 4; return;

Name: Sendistop . **Fig: 9**
Admitted in: N NE SA CA
Input: -
Requires: Slowspeed and speed = 0, pos = 1
If branch: No **Condition: -**
Effect: Vehicle has stopped behind vehicle ahead. No maystop sent Breakdown ahead?

1. If no contact, send message I-stop to system, with loc of vehicle ahead;
2. Else send collision;
3. Endif; return;

Name: Slowspeed **Fig: 9**
Admitted in: N NE SA CA
Input: Speed reading, distance to next vehicle
Requires: Newfalt
If branch: Yes **Condition:** Pos?
Effect:

1. If pos = 1, send message slow-speed to system; (this includes speed and distance to veh ahead) [System may respond with Maystop, a degraded mode or a setfault]
2. If speed = 0; call SendIstop;
3. Endif; endif; return;

Name: Sendstopit **Fig: 9**
Admitted in: N NE SA CA
Input: -
Requires: Qtwofaults
If branch: Yes **Condition:** Fault #8?
Effect:

1. Vehicle has a dangerous combination of two faults or 3 or more. Maxspeed = 0;
2. If fault #8, send message I-stop to gate, indicating loc and ln# and fault-in-me; (System will send Stopcall, inducing Stop mode)
3. Else send I-stop (same message) to system;
4. Endif; return;

A.9 NEW ACTIONS (Figure A.lO)

Name: Actcalled **Fig:** 1, 10, 12
Specification in: SUP (1)

Name: Buslead 1 **Fig:** 5, 10, 11
Specification in: Forsplit (5)

Name: Buslead4 **Fig:** 5, 10
Specification in: Forsplit (5)

Name: Callchangelane Fig: 4, 10
Specification in: Change-lane (4)

Name: Callmerge **Fig:** 2, 10
Specification in: Merge (2)

Name: Callsplit **Fig:** 3, 10
Specification in: Split (3)

Name: Emerchcall **Fig:** 6, 10
Specification in: Emer_change (6)

Name: Fault4or5 **Fig:** 10
Admitted in: N NE SA CA
Input: -
Requires: Faultset, fault #4 or fault #5 present
If branch: Yes **Condition:** Pos?
Effect: 1. If pos = last call Forspla;
2. Else call Sendaheada;
3. Endif; return;

Name: Forspla **Fig:** 5, 10
Specification in: Forsplit (5)

Name: Forspll **Fig:** 5, 10
Specification in: Forsplit (5)

Name: Fs 1 set **Fig:** 5, 10, 11
Specification in: Forsplit (5)

Name: Fs3set **Fig:** 5, 10, 11
Specification in: Forsplit (5)

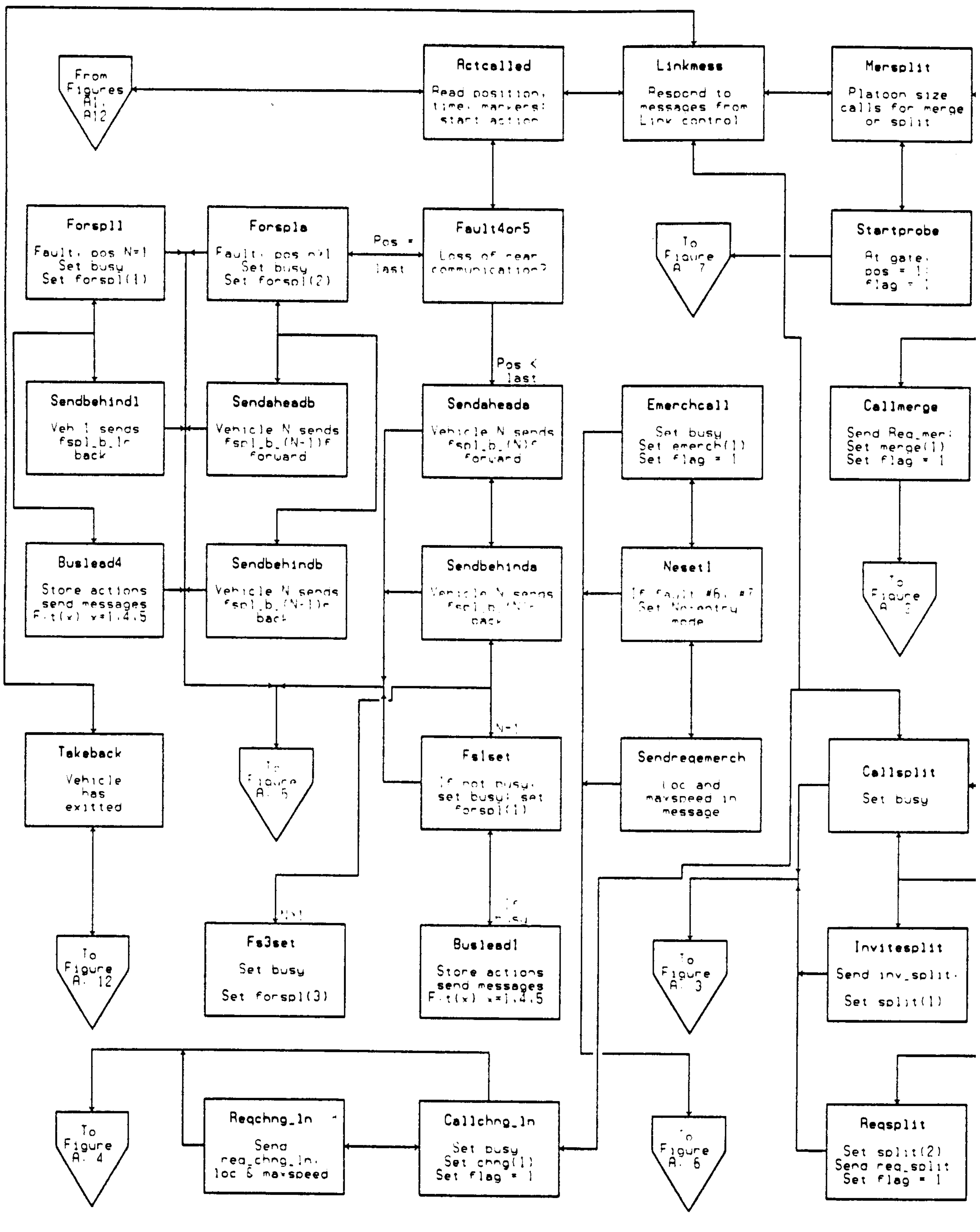


Figure A.10. Action Calls.

Name: Invitesplit . **Fig: 3,** 10
Specification in: Split (3)

Name: Linkmess **Fig:** 10
Admitted in: N NE SA CA
Input: Messages from Link Control
Requires: Faultset
If branch: 'Yes **Condition:** Any relevant messages? Free agent? Busy?

Effect:
```
1. If busy, ;
2. Else if Ln# = 0, then if message indicates enter; call Callchng_ln;
3. Else call Takeback;
4. Endif; endif; if ln# = 1 and message indicates exit at next gate and
   ownsize = 1, call Callchng_ln;
5. Else if message indicates exit at passed gate send passed-exit;
6. Endif; endif; if (time-stime) < 0, stime = time;
7. Endif; if (time - stime) > 15 sec (say), then
8. If ownsize = 1, then if message indicates change of n lanes to some loc = loc0
   and loc > loc0 - n*X,
9. Call Callchng_ln;
10. Endif; else if pos = 1 or pos = ownsize, then if message indicates change of n
    lanes to some loc = loc0 and loc > loc0 - n*X - Y,
11. Call Callsplit;
12. Endif; else if message indicates change of n lanes to some loc = loc0 and
    loc > loc0 - n*X - 2*Y,
13. Call Callsplit;
14. Endif; endif; else call Mersplit;
15. Endif; endif; return;
```

Note Recording and processing of link messages is beyond scope of this specification. X is distance taken to change lanes. It is set by Link Control on entry (will be a function of flow): Y is distance to split - set in same way.

Name: Mersplit **Fig:** 10
Admitted in: N NE SA CA
Input: -
Requires: Linkmess and not busy
If branch: Yes **Condition:** Pos = 1? Ownsize? Flag? Time?
Effect:

```
1. If pos = 1, then if flag set, then if (time - stime) excessive (30 sec?),
2. Set stime = time; if ownsize c = MO and forward probe indicates vehicle
   in range,
3. Reset flag; Call Callmerge;
4. Endif; If ownsize > = SO, reset flag; call Invite-split for vehicle with pos
   = ownsize/2;
5. Endif; endif; else set flag; set stime = time; call Startprobe;
6. Endif; endif; return;
```

Note MO - Maximum size at which platoon will seek merge - and SO - Minimum size at which platoon will split - are constants set by Link Control.

Name: Neset 1 **Fig:** 5, 10
Specification in: Emerch_change (6)

Name: Reqsplit **Fig:** 3, 10
Specification in: Split (3)

Name: Reqchngln **Fig:** 4, 10
Specification in: Change-lane (4)

Name: Sendaheada **Fig:** 5, 10, 11
Specification in: Forsplit (5)

Name: Sendbehindb **Fig:** 5, 10
Specification in: Forsplit (5)

Name: Sendbehindl **Fig:** 5, 10
Specification in: Forsplit (5)

Name: Sendreqemerch **Fig:** 5, 10
Specification in: Emer_change (6)

Name: Startprobe **Fig:** 7, 10
Specification in: Forprob (7)

Name: Takeback **Fig:** 10, 12
Specification in: Join and Quit (12)

A.10 CONTROL DATA (Figure A.ll)

Name: Buslead1 **Fig: 5,** 10, 11
Specification in: Forsplit (5)

Name: Controlmess **Fig:** 11
Admitted in: N NE SA CA
Input: Flags 1 (control data) and 2 (ack_cont) from Contdat
Requires: Contdat
If branch: Yes **Condition:** Flags
Effect: 1. If flag 1 is set, reset flag 1; set counter in Nocontmess to M; call Newvector;
2. Else call Nocontmess;
3. Call Recackcont (passing flags 1 and 2 as input);
4. Return;

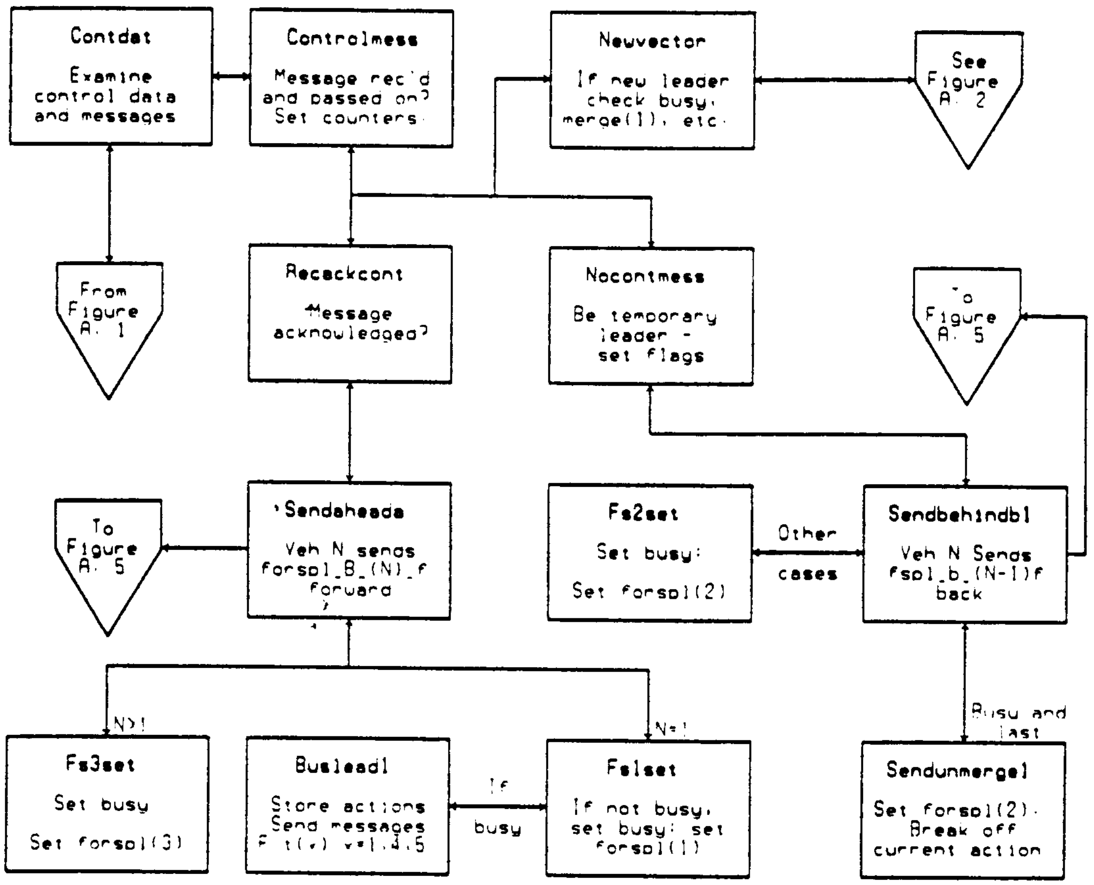
Name: Contdat **Fig:** 1, 11
Specification in: SUP (1)

Name: Fs 1 set **Fig: 5, 10,** 11
Specification in: Forsplit (5)

Name: Fs2set **Fig: 5,** 11
Specification in: Forsplit (5)

Name: Fs3set **Fig: 5,** 10, 11
Specification in: Forsplit (5)

Name: Newvector **Fig:** 11
Admitted in: N NE SA CA
Input: Control message
Requires: Controlmess
If branch: Yes **Condition:** Change of leader?
Effect: 1. If message gives leader or ownsize different from state vector, then if busy,
2. If merge(l) or merge(3) is set, reset it; reset busy; update state vector;
3. Else if split(2) is set ;
4. Else if flag A is set, ; decrement A-counter; if A-counter = 0, then
5. Reset flag A; set A counter to M; Message odd-change to system; reset busy; update state vector;
6. Endif; else set flag A; set A-counter to M;
7. Endif; endif; endif; endif; else if flag B is set, decrement B-counter; if B-counter = 0, then
8. Reset flag B; set B-counter to M; update state vector;
9. Endif; else set flag B; set B-counter to M;
10. Endif; else reset flag A; set A counter to M; reset flag B; set B-counter to M;
11. Endif; return;



Contdat
Examine control data and messages
Controlmess
Message rec'd and passed on? Set counters.
Neuvector
If new leader check busy, merge(1), etc.
See Figure A. 2
From Figure A. 1
Recackcont
Message acknowledged?
Nocontmess
Be temporary leader - set flags
To Figure A. 5
To Figure A. 5
Sendaheada
Veh N sends forsol_B_(N)_f forward
Fs2set
Set busy; Set forsol(2)
Other cases
Sendbehindbl
Veh N Sends fsol_b_(N-1)f back
N>1
N=1
Busy and last
Fs3set
Set busy Set forsol(3)
Buslead1
Store actions Send messages F t(v) v=1,4,5
If busy
Fs1set
If not busy, set busy; set forsol(1)
Sendunmerge1
Set forsol(2). Break off current action

Figure A.11. Control Data.

Name: Nocontmess **Fig:** 11
Admitted in: N NE SA CA
Input: -
Requires: Contmess
If branch: Yes **Condition:** Flags? Counter?
Effect:
```
1. If Newlead flag is set; send control data as leader; decrement counter; if
counter = 0, then
2. Reset newlead flag; set counter = M; call Sendbehindbl;
3. Endif; else decrement counter; if counter = 0, then
4. Send control data as leader; set newlead flag; set counter = M;
5. Endif; endif; return;
```

Name: Recackcont **Fig:** 11
Admitted in: N NE SA CA
Input: Flags
Requires: Controlmess
If branch: Yes **Condition:** Flag? Counter?
Effect:
```
1. If flag 2 passed is set, reset flag X; set counter to 2;
2. Else if flag 1 is set, decrement counter X;    if counter = 0, then
3. Reset flag X; set counter = 2; call Sendaheada;
4. Endif; endif; return;
```

Name: Sendaheada **Fig: 5,** 10, 11
Specification in: Forsplit (5)

Name: Sendbehindb 1 **Fig: 5,** 11
Specification in: Forsplit (5)

Name: Sendunmerge1 **Fig: 5,** 11
Specification in: Forsplit (5)

A.11 JOINING AND LEAVING (Figure A.12)

Name: Actcalled **Fig:** 1, 10, 12
Specification in: SUP (1)

Name: Askin **Fig: 12**
Admitted in: Entry lane
Input: Driver request
Requires: None!
If branch: Yes **Condition:** Destination specified?
Effect:
```
1. If destination specified, send request-enter to system;
  2. While true,
     a. examine message flags; if any set,
     b. if it is Are_you_fitted, reset flag; break;
     c. endif; else reset flag;
     d. endif;
  3. Endwhile; Go to Monsys;
  4. Else display to driver "No destination stated";
  5. Endif; stop;
```

Name: Manualok **Fig:** 1, 12
Admitted in: Exit lane
Input: -
Requires: Takeback
If branch: Yes **Condition:** Message Manual-OK received?
Effect:
```
1. If message manual-ok received,
  2. Message to driver - "have a good day!"; autocontrol off; stop;
  3. Else call Toofar;
  4. Endif; return;
```

Name: Monsys **Fig: 12**
Admitted in: Entry lane
Input: Are_you_fitted rec'd
Requires: Askin
If branch: Yes **Condition:** Controls fit?
Effect:
```
1. If MON indicates controls fit, send Im_OK; go to Waitentry;
  2. Else send Im_off; go to Nogol;
  3. Endif;
```

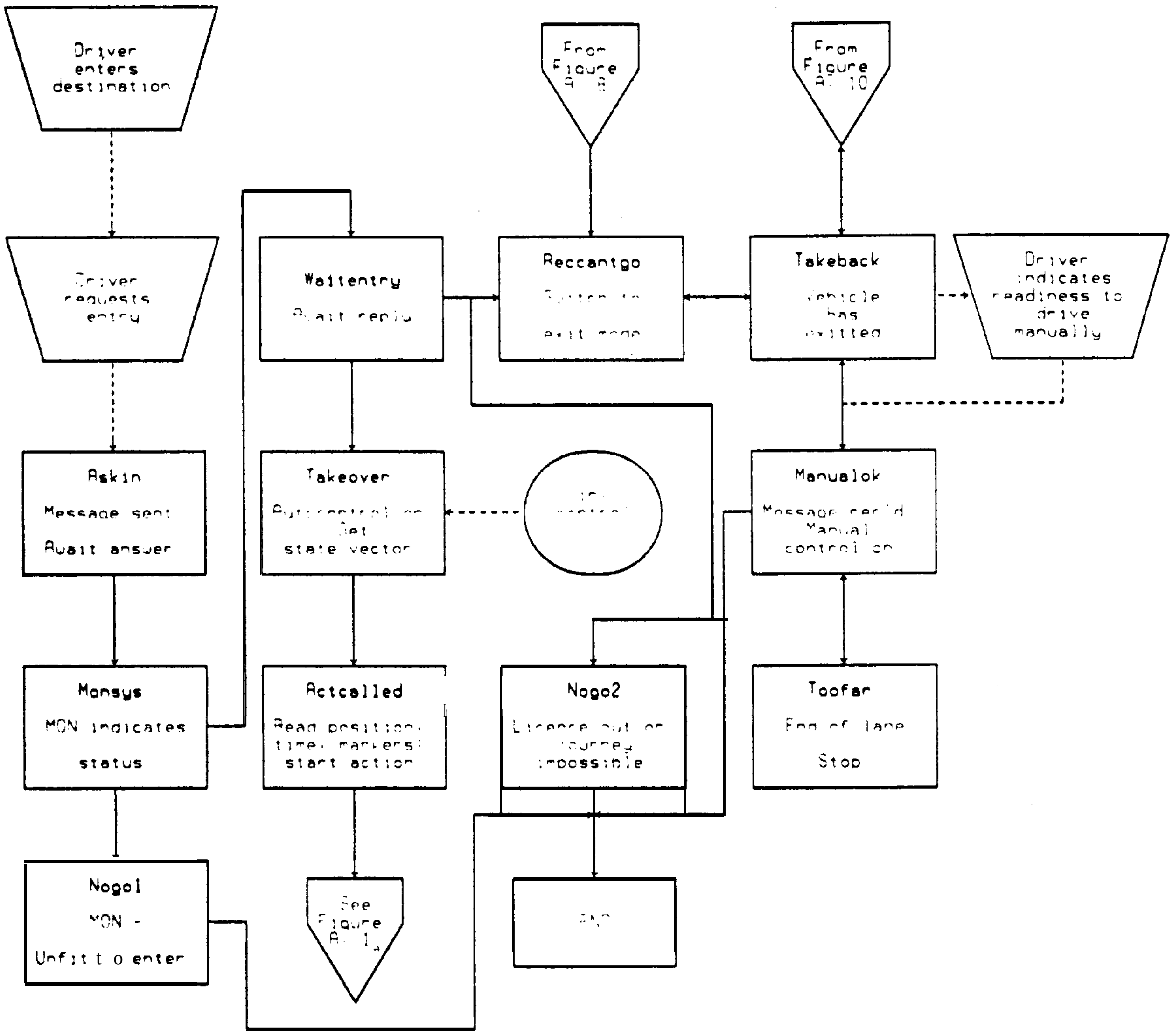

Driver enters destination
Driver requests entry
Askin Message sent Await answer
Monsys MON indicates status
Nogo1 MON - Unfit to enter
Waitentry Await reply
Takeover Autocontrol on Get state vector
Actcalled Read position, time, markers; start action
See Figure A. 14
From Figure A. 8
Reccantgo System to exit mode
Nogo2 Licence out or journey impossible
END
From Figure A. 10
Takeback Vehicle has exitted
Driver indicates readiness to drive manually
Manualok Message rec'd Manual control on
Toofar End of lane Stop

Figure A.12. Joining and Leaving.

Name: Nogol **Fig: 12**
Admitted in: Entry lane
Input: -
Requires: Monsys - controls unfit
If branch: No **Condition: -**
Effect: Display to driver "Self monitor indicates fault(S) #x, get vehicle repaired"; stop;

Name: Nogo2 **Fig: 12**
Admitted in: Entry lane
Input: Text with message no-entry
Requires: Waitentry - no-entry received
If branch: No **Condition: -**
Effect: Display to driver "Entry refused - (message text}; stop;

Note Reason may be in vehicle - license outdated, carrying external load, etc or within system - downstream section closed, destination in other direction, etc.

Name: Reccantgo **Fig: 8, 12**
Admitted in: Entry lane
Input: Message cant_go received, with gate ID
Requires: Messrec or Waitentry
If branch: No **Condition: -**
Effect: Alter link data to indicate exit at this gate; set flag in takeback; send I-quit; return;

Name: Takeback **Fig:** 10, 13
Admitted in: N NE **SA CA**
Input: -
Requires: Linkmess (10) or Reccantgo (13)
If branch: Yes **Condition:** Flag
Effect: 1. If flag set, display to Driver: "Sorry - entry barred";
2. Endif; Display to driver "Please take control"; call Manualok; (driver can send Im_ready}
3. Return;

Name: Takeover **Fig:** 1, 12
Admitted in: Entrylane
Input: Message You-are-in received
Requires: Waitentry
If branch: No **Condition: -**
Effect: 1. Set link data included in message; set state vector (from message); reset flag in Takeback;
2. Go to Actcalled;

Name: Toofar **Fig: 12**
Admitted in: Exit lane
Input: -
Requires: Manualok
If branch: Yes **Condition:** Messages received?

```
Effect: 1. If any message received,
        2. On message,

              Manual-ok:          break;
              Inch-on  :          maxspeed very low; reset flag; break;
              End-of-lane:        stop; reset flag; break;
              {default):          reset flag;
        3. Endif; return;
```

Name: Waitentry **Fig: 12**
Admitted in: Entry lane
Input: -
Requires: Monsys
If branch: Yes **Condition:** Message received

```
Effect: 1. While true,
              a. examine message flags; if any set,
              b. on flag,
                 You-are-in:      go to Takeover;
                 No-entry (with text):    reset flag; go to Nogo2;
                 Cantgo: reset flag; go to Reccantgo;
                 (default):       reset flag;
              c. endif;
        2. Endwhile;
```

A.12 SYSTEM

Note: This description does not explain how the platoon-level roadside system works. It will in fact cycle steadily, and respond to many messages from and to many vehicles. Here we consider the interaction with each vehicle separately.

The interaction takes the form of receipt of a message and a response, which may or may not require consultation of stored data.

All the following messages are received and retransmitted with the marker indicating fault #6 or #7 in the vehicle making the lane-change. Each is simply retransmitted.

Message: Request-emer-change (Sendreqemerch, fig 6)
Respond with: Request_emer_change;

Message: Ack_request_emer_change (Sendackrqemch, fig 6)
Respond with: Ack_request_emer_change;

Message: Nack_request_emer_change (Sendnackrlemch, fig 6)
Respond with: Nack_request_emer_change;

Message: Confirm_emer_change (Sendconemerch, fig 6)
Respond with: Confirm-emer-change;

Message: Thanx_but_no (Recackrqemch, recnackrqemch, fig 6)
Respond with: Thanx_but_no;

Message: Unconfirm_emer_change (Callstop, fig 6)
Respond with: Unconfirm_emer_change;

Message: Emerch_to_void (Voidemerch, fig 6)
Respond with: Emerch_to_void; set turning-point;

Message: Emer_change_at_gate_X (Qallrec, fig 6)
Respond with: Emer_change_at_gate_X; set turning-point;

Message: Adj_ahead_ok etc (Inposn, fig 6)
Respond with: Adj_ahead_ok;

The following messages cause the reactions stated in the descriptions referred to:

Messages:
Nesetl (Nesetl, fig 6) No-entry mode, if fault #6 or #7 for next two sections
Neset2 (Neset2, fig 6) No entry mode extended as gate passed
Neset3 (Neset3, fig 6) No-entry mode transformed following lane-change, and SA mode also removed.

Message: SAcall (Sacall, fig 6)
Not restricted to a particular fault. Calls SA mode to enable a safe emer-change.

Message: Fault-xqresent (Actcalled, fig 10)
Respond with: Add vehicle to list of faulty vehicles; if a vehicle remains on this list too long, advise system operators.

Message: Fault-out (Passemch, fig 6)
Respond with: Remove vehicle from list; if system operators advised of its presence, advise again;

Message: Fault_1_lveh (Forprob, fig 7).
Respond with: If another vehicle with fault #4 or #5 is present send No-fault-l; else no action;

Gate messages.

The following messages do not provoke a transmitted message but what is said below:

Messages:
Confirm_split_change_lane (in transmitted form)(Recconsplit, fig 4)
Confirm-dropt (Sendcondropt, fig 4)
Confirm-decel (Sendcondecel, fig4)
Change-to-void (Voidch, fig 4)
Emerch_to_void (Voidemerch, fig 6)
Emerch_at_gate_x (Qallrec, fig 6)

Respond with:

```
1. While 8 secs pass,
2. While giving-side VPD has not yet been occupied or is occupied,
3. If receiving side VPD is not occupied, activate turning point;
4. Else deactivate turning-point;
5. Endif; endwhile; deactivate turning point;
6. Endwhile;
```

Note. In the case of an exit gate, the receiving-side VPD extends downstream.

Entry and Exit Messages.

Message: Request-entry. (Askin, fig 12)
Respond with: Are_you_fitted to vehicle. Record existence, destination.

Message: Im_OK (MONsys, fig 12)
Respond with:
1. If licence valid and destination achievable, send Urin.
2. Else send No-entry, with reason as text; delete record of vehicle;
3. Endif;

Message: Im_off (MONsys, fig 12)
Respond with: - . Delete record of vehicle.

Message: [Vehicle has passed last on-gate]
Respond with: [Localized message] Cant-go

Message: I-quit (Reccantgo, fig 12)
Respond with: - . Delete record of vehicle

Message: [Vehicle almost at end of TL after last on-gate](Toofar, fig 12)
Respond with: [Localized message] Inch-on

Message: [Vehicle at end of TL] (Toofar, fig 12)
Respond with: [Localised message] End-of-lane

Message: Im_ready (Takeback, fig 12)
Respond with:
1. While brakes are being applied,;
2. Endwhile; send Manualok; delete vehicle from record;

Mode Changes.

All these messages are followed by advice to system operators, who alone can reverse their effects.

Messages: Crashstopbehind (Qfalt 1, fig 5), *also* unnamed messages indicating fence breach, or intrusion of platoon crush on to detector at gate.
Respond with: Crashstopcall (see Gen) for vehicles in platoon calling, plus two following sections; Cacall for following (N) sections; SAcall2 for parallel sections on lanes adjacent to CA section; advise system supervisors;
Messages: Callstop (Callstop, fig 6), I-stop (SendIstop, fig. 9)
Collision (SendIstop, fig.9), Stopit (Sendstopit, fig.9)
Respond with: Stopcall (see Gen) for two sections behind caller;

Cacall for following (N) sections; SAcall2 for parallel sections on lanes adjacent to CA sections; advise system supervisors;

Message: Slowspeed (Slowspeed, fig. 9),
Respond with: 1. If other reports in same area send May-stop to affected lanes; advise supervisors;
2. Else store info;
3. Endif;

Message: Unnamed message indicating slow-moving or stationary vehicle on exit lane, downstream of primary detector VPD.
Respond with: Sacall2 for exit lane

Suspect vehicles list.

Messages:
Norepchll (Norepchll , fig 4)
Norepchi2 (Norepchl2, fig 4)
Norepchl3 (Norepchl3, fig 4)
Norepchl4 (Norepchl4, fig 4)
Norepchl5 (Norepchl5, fig 4)
No_tp_1 (Syscalll, fig 4)
No_tp_2 (Syscall2, fig 4)
Change-messed-up (Syscall3, fig 4)
Too-long (Toolong, figs 2,3)
Respond with: 1. If there is a recently reported faulty vehicle in the same area, ;
2. Else if one of the vehicles concerned has been involved in such incidents before,
3. Advise supervisors; Send Setfalt#ll;
4. Else if gate (not for Too-long) has been involved before,
5. Close gate; advise supervisors;
6. Else add vehicles involved to suspect vehicles list;
7. Endif; endif; endif; return;

A.13 LIST OF MESSAGES

The list of messages below gives the references to figures (6 = Figure A.6, etc), in which messages are referred to.

up	=	message transmitted forward in platoon
Down	=	message transmitted rearward in platoon
out	=	message transmitted between platoons
Ex	=	message transmitted to system
In	=	message transmitted by system
*	=	message is not addressed to particular vehicle

Message	*Figure*	*Link*
Acknowledge-control-data	11	Up
Ack_request_change_lane	4	out
Ack_request_emerch	6	out
Ack_request_merge	2	Out,Down
Ack_request_split	3	Down
Adj_ahead_OK	6	out
Adj_beh_OK	6	out
Cacall	8	In
Callstop	5, 6	Ex
Cant_go	12	In
Change-mixed-up	4	Ex
Change-to-void	4	Out,Ex
Collision	9	out
Confirm-change-lane	4	out
Confirm-decelerate	4	Out,Ex
Confirm-dropt	4	Out,Ex
Confirm-emerch	6	Out,Ex
Confirm-merge	2	Out,Up
Confirm-split	3	out
Confirm_split_change_lane	4	Out,Ex
Controldata	11	Down
Crashstop	Many	Ex,In
Crashstopcall	8	In
Cutspeed	8	In
Emerch_at_gate_x	6	Out.Ex
Emerch_target	6	Ex
Emerch_to_void	6	Out,Ex
Fault-lgrobe	9	out*
Fault-xgresent	9	Ex
Forced-split-called	4,5	out

Message	*Figure*	*Link*
Forsplit_b_(N)_f	5,7	Up
Forsplit_b_(N)_r	5,7	Down
Forsplit_over	7	Down
Gatepassed	6	Ex
Im_off	12	Ex
Im_OK	12	Ex
Im_ready	12	Ex
Inch-on	12	Ex
Invite-split	3	Down
I-quit	12	Ex
I-stop	9	Ex
Maystop	8	In
Merge_3_set	2	Down
Nack_request_change_lane	4	out
Nack_request_emerch	6	out
Nack_request_merge	2	Out,Down
Nack_request_split	3	Down
Necall	8	In
Neset 1	6	Ex
Neset2	6	Ex
Neset3	6	Ex
Next-ahead-OK	6	out
Next_beh_OK	6	out
No-fault- 1	7	In
Norcall	8	In
Norepchl 1	4	Out,Ex
Norepchl2	4	Out,Ex
Norepchl3	4	Out,Ex
Norepchl4	4	Out,Ex
Norepchl5	4	Out,Ex
No_tp_1	4	Out,Ex
No_tp_2	4	Out,Ex
Probe- 1	7	out*
Probe-2	7	out*
Probe-nov	7	out*
Request-change-lane	4	out*
Request-decelerate	4	out
Request-emerch	6	out*
Request-entry	12	Ex
Request-merge	2	Out*,Up
Request-split	3	Up
Request-split-change-lane	3, 4	Down

Message	*Figure*	*Link*
Sacall	6	Ex
Sacall2	8	In
Slowspeed	9	Ex
Split-over	3	Down
s topcall	8	In
Stopit	9	Ex
Thanx_but_no	496	out
Toofar	12	In
Toolong	2,3,4	Ex
Unconfirm-change-lane	4	out
Unconfirm_emerch	6	out
Unmerge	2,5	out

7000\7400\specif