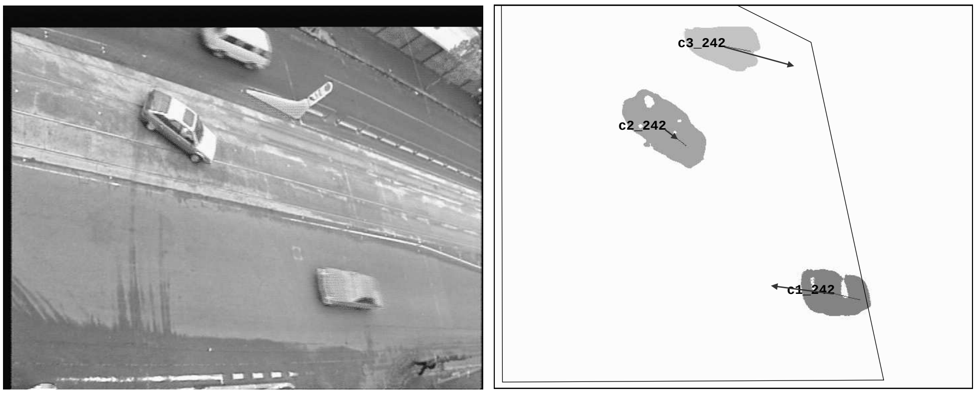
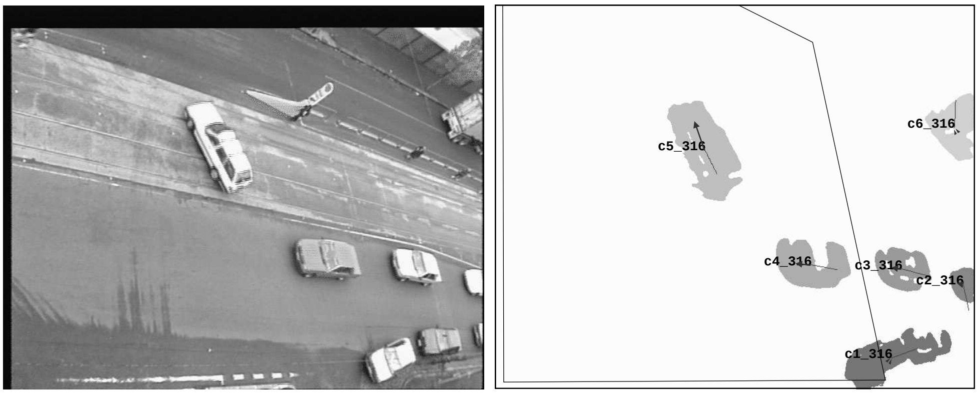


(a) (b)

Figure 9.10: *Swanston & Faraday Sts., frame 242. (a) Cars in intersection, original image. (b) Cars found by system, with orientations given by lines through cars, and their labels. Intersection boundary input manually, north is up. The interpretation follows:*

*Give Way to oncoming: id(car,c2_242,bel(1,1)) turning right from west should have (but hasn't) given way to id(car,c1_242,bel(1,1)) from east : bel(0.40,0.97)



(a) (b)

Figure 9.11: *Swanston & Faraday Sts., frame 316. (a) Cars in intersection, original image. (b) Cars found by system, with orientations given by lines through cars, and their labels. Intersection boundary input manually, north is up. The interpretation follows:*
*Illegal: car id(car,c5_316,bel(1,1)) from east is doing a right turn on the wrong side of id(tInXn,1,bel) : bel(0.72,0.78)

Chapter 10

Conclusion

The philosopher Dan Dennett has described in [25] his concept of how consciousness is a product of a network of interacting agents, none of which is the "center" or "seat" of consciousness. In this book we have demonstrated a system for interpreting images which, likewise, has no central controller. Rather it is a network of interacting (peer) agents, which, nevertheless, produces stories about images. This is a far cry, of course, from consciousness, but it does demonstrate how interesting high-level concepts can be generated from low-level data through networks of agents. In this book, the agents (concept-frames, or in Hewitt's language, actors) behave as Hewitt prescribed [38]. The relationship with Minsky's frames [47], is also close, but his slots (or terminals) are not explicitly implemented in SOO-PIN, their role is taken by property lists attached to each concept-instance. Together with the high-level logic programming language in which SOO-PIN is implemented, property lists provide a flexibility well suited for building a wide variety of concept-frames necessary in a rich domain. At an engineering level, the SOO-PIN system has demonstrated that an Object-Oriented Concurrent Logic Programming approach is a viable means of producing useful image interpretation systems. Another capability shown to be feasible by SOO-PIN is the use of a form of Dempster-Shafer uncertainty calculus, which was shown to convey useful uncertainty values through the network and produce intuitively reasonable uncertainty values at the final interpretation stage.

At the level of the traffic scenario, the system generated reasonable high level interpretations which would be suitable for input into a number of systems, for instance, quite sophisticated traffic intersection statistics could be accumulated, which would be of use to highway engineers. It would not take too much extension to connect such a traffic interpretation system to actual traffic light control and adjust the lights as a function of the types of driving behaviors detected. In fact, in conjunction with a street-level camera, the system could detect and report more complex illegal movements of cars as part of a traffic law infringement system.

Clearly, the present traffic implementation has many limitations. For instance, the background subtraction technique for low-level processing is crude, and results in problems with shadows and obscuration by overhanging structures, and limits the obliqueness of the view to the near vertical (see Figures 9.5 9.6 9.11). The system would be more robust with an intelligent object recognition system, such as one based on evidence-based techniques [13][8]. Output from such low-level processors should include uncertainty values which could be incorporated with the uncertainty measures currently propagated through SOO-PIN.

Improvements to the traffic implementation could also be made in the range of concept-frames. For instance, the system was not developed to detect whether a car was *intending* to turn left or right while still in the road next to an intersection. Also, the `whyStopped` concept-frame could be extended to reason that cars stop because the car in front is blocking the way. The system could also be extended to deal with other legally significant entities like pedestrians, trams and emergency vehicles.

The uncertainty measure propagated through the system could be improved by making the system more reactive to it, for instance, eliminating hypotheses that have a belief below a certain threshold. The `belUpd` message is a useful means of dealing with negative information (ie, the deletion of concept-instances) and of updating belief values around the network. This could have been implemented for the traffic scenario, except that, in some ways, the traffic scenario was quite simple, and there was no situations where hypotheses need to be deleted. Another improvement would be to map the uncertainty values onto English phrases (hedges) for output, and it would be an interesting problem to determine the most reasonable English phrases for the various combinations of support and plausibility output by the system.

Future work for the SOO-PIN system also includes extending the network of frames into low-level processing, creating concept-frames concerned with segments, image attributes, labeled regions and objects. With such a system, it would be possible to re-segment portions of the image where high-level concept-frames expect to find objects (ie, cars) while varying the segmentation parameters. This would be similar to the techniques used in Schema [27], Sigma [45] and that of Bell and Pau [7].

In the traffic scenario, SOO-PIN was working on sparse images derived from video image sequences. An obvious extension would be to deal with the sequence in its entirety. This would require a fundamental rethink of the architecture of the system, involving concepts of short-term and long-term memory, time-varying versus static objects, and a different form of output (whereas currently the system outputs interpretations when it has finished running, there would need to some way of determining when was the right time to generate output if it was running over a long sequence).

Finally, for the programmer, a useful improvement would be a graphical user interface (GUI) for building the system, similar to the system employed by Garvey [31] for his belief network. The current system was built using the high-level logic programming language Parlog++, which greatly facilitated the process, but it would be good to avoid the need for future programmers to learn this language. If there was a suitable GUI in which the logical nature and relationships of each concept-frame could be defined, then the Parlog++ code could be compiled from it. More speculatively, and in keeping with Cognitive Science, it would be a significant development if Machine-Learning could be involved in the construction (or evolution) of the network of concept-frames.

Appendix A

Parlog++ Procedures

In this appendix we list selected code in the Parlog++ language used in SOOPIN.

A.1 Switchboard Source Code

The first source code example is the switchboard which controls message flow around the network, and spawns new concept-frames as needed.

```
/* PARLOG++ 'switchboard' that accepts messages on the input stream
   and routes them to the appropriate concept-frame.  If not found,
   it spawns the addressed concept-frame.  */

switch.
        InStr istream, WriStr ostream

invisible ProcList state <= [], ObjType state <= switch,
        MsgCount state <= 0

clauses

        InStr::msg(NameTo,Msg) =>
                find_name(ProcList,NameTo,ProcInput,PLshort):
                %debug, inq next line
                ProcInput = [Msg|ProcInput1]&
                WriStr::msg(NameTo,Msg)&        /*debug*/
                %writeMy(OutFile,msg(NameTo,Msg))&
                Count is MsgCount + 1,
```

```
        writeCount(Count)&
        MsgCount becomes Count&
        ProcList becomes [proc(NameTo,ProcInput1)|PLshort] ;

InStr::msg(NameTo,Msg) =>
        checkProc(NameTo):
        Proc =.. [NameTo,[Msg|NewProcInput],NewProcOutput],
        call(Proc),        /*parallel, spin off new process*/
        WriStr::msg(NameTo,Msg),        /*debug*/
        %writeMy(OutFile,msg(NameTo,Msg))&

        merge(NewProcOutput,InStr,NewInStr),
        ProcList becomes [proc(NameTo,NewProcInput)|ProcList],
        Count is MsgCount + 1,
        writeCount(Count)&
        MsgCount becomes Count&
        InStr becomes NewInStr ;

InStr::msg(NameTo,Msg) =>
        Count is MsgCount + 1,
        writeCount(Count)&
        MsgCount becomes Count&
        WriStr::[unknown_proc,NameTo,Msg]&& ;

InStr::err(ErrMsg) =>
        Count is MsgCount + 1,
        writeCount(Count)&
        MsgCount becomes Count&
        WriStr::[error_rcvd,ErrMsg]&& ;

InStr::info(InfoMsg) =>
        Count is MsgCount + 1,
        writeCount(Count)&
        MsgCount becomes Count&
        WriStr::[info_rcvd,InfoMsg]&& ;

InStr::draw =>
        drawNet(ProcList,NewProcList)&
        WriStr::draw&
        Count is MsgCount + 1,
        writeCount(Count)&
        MsgCount becomes Count&
        ProcList becomes NewProcList&& ;

InStr::quit =>
        WriStr::stopping_procs&
        WriStr::msgCount(MsgCount)&
```

```
                stop_proc(ProcList)&& ;

        InStr::last =>
                WriStr::last&& ;

        InStr::BadMsg =>
                WriStr::[bad_message,BadMsg].
code

mode find_name(?,?,^,^).

        /*find process named, outputting its input stream and a
          proc list without that process. If not found then fail*/
find_name([proc(Name,ProcInput)|PLshort],Name,ProcInput,PLshort) <-
        true;
find_name([Proc|PLRest],Name,ProcInput,PLshort) <-
        PLshort = [Proc|PLshorter] &
        find_name(PLRest,Name,ProcInput,PLshorter).

mode drawNet(?,^).

drawNet(ProcList,NewProcList) <-
        open(network,write,NetFile)&
        drawProc(ProcList,NetFile,NewProcList)&
        close(NetFile)&
        display_network.

mode drawProc(?,?,^).

drawProc([],_,[]).

drawProc([proc(Name,ProcInput)|Rest],NetFile,
                        [proc(Name,ProcInput1)|Rest1]) <-
        ProcInput = [dump(InstList,ObjLevel)|ProcInput1]&
        writeNet(NetFile,InstList,ObjLevel)&
        drawProc(Rest,NetFile,Rest1).

mode stop_proc(?).

stop_proc([]).

stop_proc([proc(Name,ProcInput)|Rest]) <-
        ProcInput = []&
        stop_proc(Rest).

mode writeMsg(?,?).
%write destination and type of message
```

```
writeMsg(OutFile,msg(NameTo,Msg)) <-
        functor(Msg,MsgType,Arity)&
        writeMy(OutFile,[NameTo,MsgType]).

mode writeCount(?).
%write count if its a multiple of 5
writeCount(Count) <-
        Rem is Count mod 5&
        Rem =:= 0:
        write('''*''')&
        flush_output(user_output);
writeCount(Count).

end.
```

A.2 Give-Way Source Code

In the next example, a *giveWay* concept-frame is shown, that which detects the existence of a give-way situation between a U-turning car and an oncoming car.

```
giveWayUt.        %give way to oncoming traffic when U-turning on road
        Out ostream
invisible InstList state <= [], ObjType state <= giveWayUt,
                        ObjLevel state <= 3, OutFile state
initial open(ObjType,write,OutFile)&
        writeMy(OutFile,mmmmmmmmmmmmmmmmmmmmmmmmmmmmm)&
        xterm(ObjType)&& .
clauses
        last =>        writeMy(OutFile,[stopping,InstList])&
                close(OutFile)&
                Out::last.      /*dump insts & stop*/

        dump(OutInstList,OutObjLevel)          =>
                OutInstList = InstList&
                OutObjLevel = ObjLevel && ;

        create(Id,Relns)        =>
                getInst(Id,InstList,inst(TargId,Props,Justn),ExcInstList):
                writeList(OutFile,[create,unnecessary,as,Id,found])&
                union(Relns,Props,NewProps)&
                InstList becomes [inst(TargId,NewProps,Justn)|ExcInstList]

                else        /*inst not found */
                writeList(OutFile,[created,Id])&
                checkAssns(Id)& /*send 'check' to likely
                                        associates with this inst Id*/
                InstList becomes
                        [inst(Id,Relns,[])|InstList] && ;
                /*dont process any more msgs until InstList is updated!*/

        check(reln(composedOf,SendId,Bel),Done) =>
                %Note Done flag to delay sender until this object updated
                Done = yes&
                writeList(OutFile,[check,relation,composedOf,from,SendId])&
                getType(SendId,SendType)&
                %get the road the 'right' is in
                Out::msg(SendType,getVal(SendId,
                        reln(composedOf,id(road,any,bel),bel),RoadProps))&
                Out::msg(SendType,getVal(SendId,
```

```
                desc(from(any),bel),RtFrom))&
        sendRoad(OutFile,InstList,RoadProps,RtFrom,SendId,
                NewInstList)&

        /*found if FoundIds not empty*/
        InstList becomes NewInstList&& ;

negCheck(Id) =>
        %Id is removed from the Property lists of all Insts
        writeList(OutFile,[Id,removed,from,all,relations])&
        delRef(InstList,Id,NewInstList)&
        InstList becomes NewInstList&& ;

anyInst(FoundList,Prop) =>
        writeList(OutFile,[enquiry,regarding,Prop])&
        /*Note: this only checks properties in the PropList*/
        seek(InstList,Prop,FoundList)&& ;

getVal(WantId,Prop,FoundPropList) =>
        %propagate to other traffic obj
        getInst(WantId,InstList,inst(InstId,Props,Justn),ExcInstList):
                /*fail if not found*/
        writeList(OutFile,[enquiry,to,WantId,re,Prop])&
          searchProps(Props,Prop,FoundPropList)&& ;

getVal(WantId,Prop,FoundPropList) =>
        writeList(OutFile,[enquiry,to,WantId,re,Prop,failed])&
        FoundPropList = []&& ;   /*If Id wrong, return []*/

updVal(Id,Relns) =>
        getInst(Id,InstList,inst(TargId,Props,Justn),ExcInstList):
        writeList(OutFile,[update,Id,with,Relns])&
        union(Relns,Props,NewProps)&
        InstList becomes [inst(TargId,NewProps,Justn)|ExcInstList]

        else    /*inst not found, so create it but dont checkAssns*/
        writeList(OutFile,[update,of,Id,with,Relns,failed,not,found])&
        InstList becomes
                [inst(Id,Relns,[])|InstList] && ;

inq =>  writeMy(OutFile,[inq,InstList])&& ;

WrongMsg =>     writeMy(OutFile,[bad_msg,WrongMsg]).
```

code

```
mode checkAssns(?,^,?).
/* check normal associations of this object type to see if they exist in the
   expected relationship.  Note the result of this checking is returned in
   msg 'updVal'  */

checkAssns(InstList,InstList,InstId).

/*************************************************************************/
mode sendProps(?,?).
/* send list of properties to another object instance */
sendProps(_,[]);        %if no props, dont send anything
sendProps(id(ToType,ToNum,Bel),Props) <-
        Out::msg(ToType,updVal(id(ToType,ToNum,Bel),Props)).

/*************************************************************************/
mode sendRoad(?,?,?,?,?,^).         %OutFile,InstList,RoadProps,
                                    %RtFrom,SendId,NewIL
%send to straight to get cars from other dirn
sendRoad(OutFile,InstList,[reln(composedOf,id(road,RoadNo,BelI),Bel)|Rest],
                [desc(from(Dir),BelD)|Rest2],SendId,NewIL) <-
        length(Rest,Len)&
        writeListCond(user_output,Len,[SendId,composed,of,
                more,than,one,inXn,Rest])&
        length(Rest2,Len2)&
        writeListCond(user_output,Len2,[SendId,from,more,
                than,one,dirn,Rest2])&
        Out::msg(straight,anyInst(FoundStr,reln(composedOf,
                id(road,RoadNo,BelI),Bel)))&
        getIdLFromInstL(FoundStr,FoundIds)& %%%ds
        checkTarget(OutFile,InstList,targetOf,FoundIds,
                        SendId,NewIL);

sendRoad(OutFile,InstList,[],_,_,InstList);         %empty composure
sendRoad(OutFile,InstList,_,[],_,InstList);         %empty from list
%error
sendRoad(OutFile,InstList,Wrong1,Wrong2,Wrong3,InstList) <-
        writeList(OutFile,[bad,call,to,sendRoad,in,giveWayUt,ie,Wrong1,
                Wrong2,Wrong3]).

/*************************************************************************/
mode checkTarget(?,?,?,?,?,^). %OutFile,InstList,TargReln,CommStr,SendId,NewIL
%before creating the Inst, check car is pointing toward subject

checkTarget(OutFile,InstList,TargReln,[Id|Rest],SendId,NewIL) <-
        Out::msg(uTurn,getVal(SendId,reln(composedOf,
                id(car,any,bel),bel),SendCarPropList))&
        getType(Id,IdType)&
```

```
        Out::msg(IdType,getVal(Id,reln(composedOf,id(car,any,bel),bel),
                        CarPropList))&
        checkTarget2(OutFile,InstList,TargReln,Id,CarPropList,
                SendId,SendCarPropList,NewIL1)&
        checkTarget(OutFile,NewIL1,TargReln,Rest,SendId,NewIL);

checkTarget(_,InstList,_,[],_,InstList).

/*************************************************************************/
mode checkTarget2(?,?,?,?,?,?,?,^).
%get the single car from the lists CarPropList & RightCarPropList, send
%to Ors to find if car coming targets left, if so call create.
%first check case where sender is target of car to give way to
checkTarget2(OutFile,InstList,targetOf,Id,[reln(composedOf,CarId,Bel)|Rest],
               SendId,[reln(composedOf,CarSendId,Bel2)|Rest1],NewIL) <-
        Out::msg(car,getVal(CarId,desc(reversed,bel),Return))&
        reverseReln(targetOf,Return,RReln)&
        checkOrs(RReln,CarSendId,CarId,FoundProps)&
        condCreateInst(OutFile,InstList,Id,CarId,SendId,CarSendId,
                FoundProps,NewIL);
%next check case where sender is targetting car to give way
checkTarget2(OutFile,InstList,targetting,Id,[reln(composedOf,CarId,Bel)|Rest],
               SendId,[reln(composedOf,CarSendId,Bel2)|Rest1],NewIL) <-
        Out::msg(car,getVal(CarSendId,desc(reversed,bel),Return))&
        reverseReln(targetting,Return,RReln)&
        checkOrs(RReln,CarSendId,CarId,FoundProps)&
        condCreateInst(OutFile,InstList,SendId,CarSendId,
                Id,CarId,FoundProps,NewIL);

checkTarget2(OutFile,InstList,_,Id,[],SendId,_,InstList) <-
        writeList(OutFile,['Error in checkTarget2, got no cars composing',
                Id]);
checkTarget2(OutFile,InstList,-,Id,_,SendId,[],InstList) <-
        writeList(OutFile,['Error in checkTarget2, got no cars composing',
                SendId]).

/*************************************************************************/
mode reverseReln(?,?,^).
%if Return from car contains desc(reversed,bel), ie is non-empty,
        %then put 'B' on the end of input reln and pass back
reverseReln(Reln,[desc(reversed,BelD)],NewReln) <-
        concatStr([Reln,'''B'''],NewReln);
reverseReln(Reln,_,Reln).

/*************************************************************************/
mode condCreateInst(?,?,?,?,?,?,?,^). %OutFile,InstList,SignDesc,
        %StraightId,StrCarId,StDir,SendId,CarSendId,SeDir,FoundProps,NewIL
```

```
%dont create if FoundProps is []
condCreateInst(_,InstList,_,_,_,_,[],InstList);
%test if new components are in existing Inst, create if not
condCreateInst(OutFile,InstList,Id,CarId,SendId,CarSendId,FoundProps,NewIL) <-
        dupeInst(InstList,[reln(composedOf,Id,bel),
        reln(composedOf,SendId,bel)],DupIds)&
        %DupIds is list of Ids of Insts with components
        DupIds =@= []:
        createInst(OutFile,InstList,Id,CarId,SendId,
                CarSendId,[reln2(dum,Id,SendId,bel)],NewIL);  %%ds
%or do nothing
condCreateInst(_,InstList,_,_,_,_,_,InstList).

/************************************************************************/
mode createInst(?,?,?,?,?,?,?,^).   %OutFile,InstList,SignDesc,StraightId,
                                    %StrCarId,StDir,SendId,CarSendId,
                                    %SeDir,Justn,NewIL
%create new insts for members of CommStr
createInst(OutFile,InstList,Id,CarId,SendId,
                        CarSendId,Justn,NewIL) <-
        getNewOrsAddr(NewNo)&
        belTwoOfN(Justn,NewBel)&
        union([inst(id(giveWayUt,NewNo,NewBel),[reln(composedOf,Id,bel),
                reln(composedOf,CarId,bel),reln(composedOf,SendId,bel),
                reln(composedOf,CarSendId,bel),
                desc(subject(CarId),bel)],Justn)],InstList,NewIL)&
        getVel(CarSendId,Vel)&
        velPhrase(Vel,Phrase)&
        name(NL,[10,13])&
        Out::msg(result,result(['''Give Way to Oncoming: ''',CarSendId,NL,
                ''' U-turning''',Phrase,to,CarId,going,
                straight,''': ''',NL,NewBel]))&
        writeList(OutFile,[adding,inst,NewNo,composed,of,Id,SendId,CarId,
                CarSendId])&
        getNewOrsAddr(NewNo1)&
        Out::msg(giveWay,create(id(giveWay,NewNo1,NewBel),
                [reln(givesWay,CarId,bel),reln(rightOfWay,CarSendId,bel)]))&
        sendProps(Id,[reln(partOf,id(giveWayUt,NewNo,NewBel),bel)])&
        sendProps(SendId,[reln(partOf,id(giveWayUt,NewNo,NewBel),bel)]).

/************************************************************************/
mode velPhrase(?,^).        %OffendId,velocity phrase
%send msg to result conditional upon speed of OffendId
velPhrase([],''' gives way ''');        %velocity unknown
velPhrase([Vel],'''has given way''') <-
        stopped(Vel,stopped):
        true;
```

```
velPhrase([Vel],'''should have (but hasn't) given way''');
velPhrase(Vel,_) <-
        writeMy([error,in,velPhrase,parameter,is,Vel])&
        fail.

/*****************************************************************************/
mode getVel(?,^). %id(Car,CarNo,Bel),[desc(vel(VelX,VelY),BelV)]
%get the velocity desc from car, if not there return []  %%vel
getVel(id(Car,CarNo,Bel3),VelList) <-
        Out::msg(Car,getVal(id(Car,CarNo,Bel3),desc(vel(any,any),bel),
                     VelList)).


end.
```

References

[1] Aisbett, J. Optical flow with an intensity-weighted smoothing. *IEEE Transactions on Pattern Analysis and Machine Intelligence 11*, 5 (May 1989).

[2] Allen, J. *Natural Language Understanding.* Benjamin/Cummings, Menlo Park, 1987.

[3] Andersen, S. K., Olesen, K. G., Jensen, F. V., and Jensen, F. HUGIN – a shell for building Bayesian belief universes for expert systems. In *IJCAI-89* (Detroit, August 1989), pp. 1080–1085.

[4] Andre, E., Herzog, G., and Rist, T. On the simultaneous interpretation of real world image sequences and their natural language descriptions: the system SOCCER. In *ECAI 88. Proceedings of the 8th european conference on artificial intelligence* (UK, 1988), Y. Kodratoff, Ed., pp. 449–54.

[5] Bajcsy, R., Joshi, A., Krotkov, E., and Zwarico, A. LANDSCAN: a natural language and computer vision system for analysing aerial images. In *IJCAI 85: Proceedings of the Ninth Intl Joint Conf on Artificial Intelligence* (1985), Int Joint conferences on Artificial Intelligence Inc, Morgan Kaufmann.

[6] Baldwin, J. Support logic programming. In *Fuzzy sets - Theory and Applications, Proceedings of NATO Advanced Study Institute*, A. Jones et al., Eds. Reidel Pub. Co., Norwell, MA, 1986.

[7] Bell, B., and Pau, L. F. Context knowledge and search control issues in object-oriented prolog-based image understanding. *Pattern Recognition Letters 13*, 4 (April 1992), 279–290.

[8] Bischof, W. F., and Caelli, T. Learning structural descriptions of patterns: a new technique for conditional clustering and rule generation. *Pattern Recognition 27*, 5 (1994), 689–697.

[9] Bobick, A. F., and Bolles, R. C. The representation space paradigm of concurrent evolving object descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 2 (Feb 1992), 146–156.

[10] Bodington, R., Sullivan, G., and Baker, K. Experiments on the use of the ATMS to label features for object recognition. In *Computer Vision - ECCV90* (Antibes, France, April 1990), O. Faugeras, Ed., vol. 427 of *Lecture Notes in Computer Science*, INRIA, Springer Verlag, pp. 542–551.

[11] Bogler, P. L. Shafer-Dempster reasoning with applications to multisensor target identification systems. *IEEE Transactions on Systems, Man and Cybernetics SMC-17*, 6 (November 1987), 968–977.

[12] Bunke, H. Hybrid methods in pattern recognition. In *Pattern Recognition Theory and Applications*, P. A. Devijer and J. Kittler, Eds., vol. F30 of *NATO ASI*. Springer-Verlag, 1987.

[13] Caelli, T., and Dreier, A. Some new techniques for evidence-based object recognition: EB-ORS1. In *IAPR-92 Proceedings* (Hague, September 1992), pp. 450–455.

[14] Charniak, E., and McDermott, D. *Introduction to Artificial Intelligence.* Addison-Wesley, Reading, Massachusetts, 1985.

[15] Cipolla, R., and Yamamoto, M. Stereoscopic tracking of bodies in motion. *Image and Vision Computing 8*, 1 (february 1990), 85–90.

[16] Clancey, W. J. Situated cognition: How representations are created and and given meaning. In *AERA 1991 Symposium* (Chicago, 1991).

[17] Conlon, T. *Programming in PARLOG.* Addison-Wesley, Menlo Park, Ca., 1989.

[18] Dance, S., and Caelli, T. On the symbolic interpretation of traffic scenes. In *ACCV93 Proceedings of the Asian Conference on Computer Vision* (Osaka Japan, november 1993), pp. 798–801.

[19] Dance, S., and Caelli, T. A symbolic object-oriented picture interpretation network: SOO-PIN. In *Advances in Structural and Syntactic Pattern Recognition, Proceedings of the International Workshop* (Bern, Switzerland, 1993), H. Bunke, Ed., World Scientific Publishing Co., pp. 530–541.

[20] Dance, S., Caelli, T., and Liu, Z.-Q. A network-of-frames model for symbolic scene interpretation. Submitted for publication to Pattern Recognition Journal, August 1994.

[21] Davison, A. From Parlog to Polka in two easy steps. In *PLILP'91 : 3rd Int. Symp. on Programming Language Implementation and LP* (Passau, Germany, August 1991), no. 528 in Springer LNCS, Springer, pp. 171–182.

[22] de Kleer, J. An assumption based TMS. *Artificial Intelligence 28*, 2 (March 1986).

[23] Dellepiane, S., Serpico, S. B., and Vernazza, G. 3D organ recognition by tomographic image analysis. In *Pattern Recognition Theory and Applications*, P. A. Devijer and J. Kittler, Eds., vol. F30 of *NATO ASI.* Springer-Verlag, 1987.

[24] Dempster, A. P. A generalization of Bayesian inference. *Journal of the Royal Statistical Society 30* (1968), 205–247.

[25] DENNETT, D. *Consciousness explained.* Little, Brown and Co., Boston, 1991.

[26] DILLON, C. Image pattern recognition system (IPRS) user manual. Tech. Rep. 93/14, Computer Science, University of Melbourne, Parkville, Victoria, Australia, 1993.

[27] DRAPER, B. A., COLLINS, R. T., BROLIO, J., HANSON, A. R., AND RISEMAN, E. M. The Schema System. *International Journal of Computer Vision 2* (1989), 209–250.

[28] DUBOIS, D., AND PRADE, H. A discussion of uncertainty handling in support logic programming. *International Journal of Intelligent Systems 5*, 1 (March 1990), 15–42.

[29] FELLER, W. *An Introduction to Probability Theory and its Applications*, 3rd ed., vol. 1. Wiley and Sons, New York, 1968.

[30] FERI, R., FORESTI, G., MURINO, V., REGAZZONI, C., AND VERNAZZA, G. Spatial reasoning by knowledge-based integration of visual and IR fuzzy cues. In *Signal Processing V. Theories and Applications. Proceedings of EUSIPCO-90, Fifth European Signal Processing Conference* (1990), L. Torres, E. Masgrau, and M. Lagunas, Eds., Elsevier Amsterdam, Netherlands, pp. 1719–22 vol.3.

[31] GARVEY, T. D. Evidential reasoning for geographic evaluation for helicopter route planning. *IEEE Transactions on Geoscience and Remote Sensing GE-25*, 3 (May 1987), 294–303.

[32] GONZALEZ, R. C., AND WINTZ, P. *Digital Image Processing.* Addison Wesley, Reading, Massachusetts, 1987.

[33] GOVINDARAJU, V., LAM, S. W., NIYOGI, D., SHER, D. B., SRIHARI, R., SRIHARI, S. N., AND WANG, D. Newspaper image understanding. In *Knowledge based computer systems*, S. Ramani, R. Chandrasekar, and K. S. R. Anjaneyalu, Eds. Narosa Publishing House, New Delhi, India, 1990, pp. 375–84.

[34] Green, P. E. AF: a framework for real-time distributed cooperative problem solving. In *Distributed Artificial Intelligence*, M. N. Huhns, Ed. Pitman, London, 1987, ch. 6, pp. 153–176.

[35] Hayes, P. J. The naive physics manifesto. In *Expert Systems in the Micro-Electronic Age*, D. Michie, Ed. Edinburgh University Press, Edinburgh, Scotland, 1979.

[36] Herskovits, A. *Language and Spatial Cognition.* Cambridge University Press, 1986.

[37] Hewitt, C. How to use what you know. In *IJCAI-75* (Tbilisi, Georgia, September 1975), pp. 189–198.

[38] Hewitt, C. Viewing control structures as patterns of passing messages. *Artificial Intelligence 8* (1977), 323–363.

[39] Hewitt, C. E. Planner: A language for proving theorems in robots. In *IJCAI'69* (Washington, D.C., 1969).

[40] Hota, N., Ramsey, C. L., Chang, L. W., and Booker, L. B. BaRT manual version 3.0. Tech. Rep. 6778, Naval Research Laboratory, US Navy, Washington DC, Feb 1991.

[41] Huang, T., Koller, D., Malik, J., Ogasawara, G., Rao, B., Russell, S., and Weber, J. Automatic symbolic traffic scene analysis using belief networks. In *Proc of AAAI-94* (Seattle, August 1994).

[42] Keller, J., Hobson, G., Wootton, J., Nafarieh, A., and Luetkemeyer, K. Fuzzy confidence measures in midlevel vision. *IEEE Transactions on Systems, Man and Cybernetics SMC-17*, 4 (july 1987), 676–683.

[43] Lakoff, G. *Women, Fire and Dangerous Things.* University of Chicago Press, 1987.

[44] Lowrance, J. D., Strat, T. M., Wesley, L. P., Garvey, T. D., Ruspini, E. H., and Wilkins, D. E. The theory, implementation and practice of evidential reasoning. SRI Project 5701, SRI International, Menlo Park, CA 94025, June 1991. Final report.

[45] Matsuyama, T., and Hwang, V. S. *SIGMA A Knowledge based aerial image understanding system.* Plenum Press, New York, 1990.

[46] McDermott, D. V., and Doyle, J. Non-monotonic logic I. *Artificial Intelligence 13*, 1,2 (1980), 41–72.

[47] Minsky, M. A framework for representing knowledge. In *The psychology of computer vision*, P. H. Winston, Ed. McGraw-Hill, New York, 1975.

[48] Mohnhaupt, M., and Neumann, B. On the use of motion concepts for top down control in traffic scenes. In *Computer Vision ECCV 90. First european conference on computer vision proceedings* (1990), O. Faugeras, Ed., pp. 598–600.

[49] Mulder, J. A., Mackworth, A. K., and Havens, W. S. Knowledge structuring and constraint satisfaction: The MAPSEE approach. Tech. Rep. 87-21, Dept. of computer science, Uni of British Columbia, Vancouver, BC, Canada V6T 1W5, June 1987.

[50] Neumann, B. Natural language description of time-varying scenes. In *Semantic Structures: advances in natural language processing*, D. L. Waltz, Ed. Lawrence Erlbaum, Hillsdale, N.J, 1989, pp. 167–207.

[51] Niemann, H., Bunke, H., Hofmann, I., Sagerer, G., Wolf, F., and Feistel, H. A knowledge based system for analysis of gated blood pool studies. *IEEE Transactions on Pattern Analysis and Machine Intelligence 7*, 3 (may 1985), 246–259.

[52] Pearl, J. *Probabilistic reasoning in intelligent systems: Networks of plausible inference.* Morgan Kaufman, San Mateo, CA, 1988.

[53] PR, C., AND R, K. Information retrieval by constrained spreading activation in semantic networks. *Information Processing and Management 23*, 4 (1987), 255–268.

[54] PROVAN, G. M. An analysis of knowledge representation schemes for high level vision. In *Computer Vision - ECCV90* (Antibes, France, April 1990), O. Faugeras, Ed., vol. 427 of *Lecture Notes in Computer Science*, INRIA, Springer Verlag, pp. 537–541.

[55] REITER, R., AND MACKWORTH, A. A logical framework for depiction and image interpretation. *Artificial Intelligence 41* (1990), 125–155.

[56] RETZ-SCHMIDT, G. Deictic and intrinsic use of spatial prepositions. In *Spatial Reasoning and Multi-Sensor Fusion* (1987), A. Kak and S. Chen, Eds., Morgan Kaufman, pp. 371–380.

[57] RICH, E., AND KNIGHT, K. *Artificial Intelligence*, 2nd ed. McGraw-Hill, New York, 1991.

[58] RINGWOOD, G. A. The dining logicians. Master's thesis, Department of Computing, Imperial College, London, 1984.

[59] ROBINSON, J. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery 12*, 1 (1965).

[60] SCHANK, R., AND ABELSON, R. *Scripts, Plans, Goals and Understanding.* Erlbaum, Hillsdale, N.J, 1977.

[61] SCHIRRA, J. R. J., BOSCH, G., SUNG, C. K., AND ZIMMERMANN, G. From image sequences to natural language: a first step toward automatic perception and description of motions. *Applied Artificial Intelligence 1*, 4 (87), 287–305.

[62] SHAPIRO, E., AND TAKEUCHI, A. Object oriented programming in concurrent prolog. In *Concurrent Prolog*, E. Shapiro, Ed., vol. 2. MIT Press, 1987, ch. 29, pp. 251–273.

[63] Slezak, P. Situated cognition: Empirical issue, paradigm shift or conceptual confusion. In *Proceedings of the Sixteenth Conference of the Cognitive Sccience Society* (Atlanta, August 1994), A. Ram and K. Eiselt, Eds., Lawrence Erlbaum, Hillsdale, New Jersey, pp. 806–811.

[64] Tropf, and Walters. An ATN for 3D recognition of solids in single images. In *Proceedings of the 8th Int'l Joint Conf. on Artificial Intelligence* (1983).

[65] Tsotsos, J. K. The complexity of perceptual search tasks. In *Proc. International Joint Conference on Artificial Intelligence* (Detroit, August 1989), N. S. Sridharan, Ed., Morgan Kaufman, pp. 1571 – 1577.

[66] Verri, A., and Poggio, T. Against quantitative optical flow. In *First International conference on Computer Vision* (1987), IEEE, pp. 171–180.

[67] Wesley, L. P. Evidential knowledge-based computer vision. Tech. Rep. 374, A. I. Centre, SRI International, Menlo Park, CA 94025, January 1986.

[68] Winston, P. *Artificial Intelligence*, second ed. Addison-Wesley, Reading, Massachusetts, 1984.

[69] Wylie Jr, C. *Introduction to Projective Geometry.* McGraw-Hill, 1970.

[70] Yonezawa, A., and Hewitt, C. Modelling distributed systems. In *IJCAI-77* (Massachusetts, August 1977), Kaufman, pp. 370–376.

Index