A Parallel Linear & UR-Derivation System[1]

Extended Abstract

Geoff Sutcliffe
The University of Western Australia
Perth, Western Australia
geoff@cs.uwa.oz.au

1 Introduction

The exponential size of the search space of the resolution procedure necessitates the use of refined derivation systems, which restrict the search space. Given some fundamental refinements, further refined derivation systems can be developed by combining several fundamental refinements in a single derivation system. The manner in which combinations can be formed is necessarily restricted by the requirement of completeness. An alternative to combining refinements in a single derivation system is to use multiple (refined) derivation systems concurrently. All the component derivation systems of a concurrent derivation system use a common input set, and thus all components can use clauses created by other components. Although hard to quantify, it is this 'cross-fertilisation' between different derivation systems which furnishes the power of concurrent derivation systems. A notable feature of the concurrent approach it is only necessary for one of the component derivation systems to be complete.

The logical progression from the concurrent approach is to the parallel use of multiple derivation systems. In a parallel derivation system the component derivation systems run independently, and communicate in some (as yet) unspecified manner. The advantage that a parallel system has over a concurrent system is that the components can (and should) run on separate processors, thus enabling them to take advantage of added computing power. This paper introduces a parallel derivation system called GLD||UR, which uses a chain format linear derivation system and a UR-derivation system in parallel.

2 The Derivation Components

Guided Linear Derivation

Guided Linear Derivation (GLD) is a chain format linear derivation system, based broadly on the Graph Construction (GC) procedure [Shostak 1976]. GLD is a complete derivation system, and has features that make it suitable for use in parallel with

[1] This research was carried out at Edith Cowan University, Perth, Western Australia

UR-derivation : • an extended unit preference strategy in all extension operations, • a unique mechanism for the reuse of derived information, including application of back and forward subsumption for all new lemma chains, • a modified consecutively bounded depth first search.

UR-derivation
The UR-derivation [Overbeek 1976] system used in GLD||UR uses the same chain representation of clauses as the GLD component. As in GLD, back and forward subsumption are applied for all new unit chains created.

3 The Parallel Implementation Environment - Prolog-Linda

The parallel aspects of GLD||UR have been implemented using Prolog-Linda [Sutcliffe 1990]. Linda is a programming framework of language-independent operators which may be injected into existing programming languages, resulting in new parallel programming languages. Linda permits cooperation between parallel processes by controlling access to a shared data structure called a *tuple space*. Manipulation of a tuple space is only possible using Linda operators. Parallel execution of programs is provided for via an operator which starts the execution of new processes. Prolog-Linda is an extension of Prolog that supports tuple spaces and the Linda operators.

In our Prolog-Linda implementations the tuple space and associated operations are implemented in a *server* process. Linda operations in *client* processes are translated into requests which are passed to the server. The requests are serviced by evaluating them as Prolog queries in the server. Requests for tuple space operations are simply queries on Prolog procedures which implement those operations. This method of servicing requests is a general mechanism, and allows any query to be passed to the server for evaluation. The method may also be used in client processes, and is in fact used extensively in GLD||UR.

Prolog-Linda has been implemented in two different environments. The first is in muProlog on a network of Sun SPARC station-1s running SunOS 4.0.3. Intermachine communication is via an Ethernet, using TCP/IP protocol. The second implementation is in Arity Prolog on a network of IBM PS/2 55SXs, using MS-DOS 3.3. Intermachine communication is via a token ring running IBM Netbios protocol.

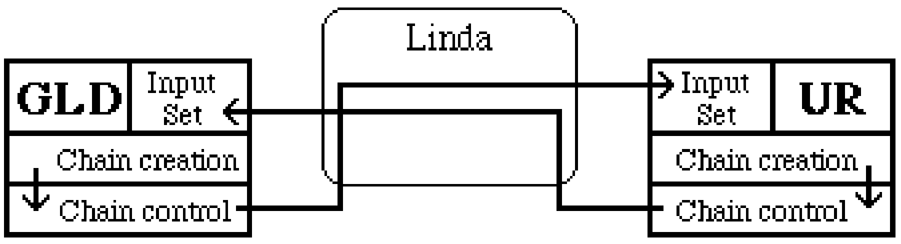
4 GLD||UR Architectures

The parallel combination of GLD and the UR-derivation system forms GLD||UR. The two systems are as described above, with extensions to implement the distribution of chains created. Both derivation components maintain their own copies of the input set, which are updated with chains that are created locally and with chains that are created in the other derivation component. In both derivation components, new chains that survive forward subsumption are added to the local input set and also transmitted to the other component. At the same time, identifiers for any chains that were locally back subsumed are also transmitted to the other component, so that they can be removed from the input set in that component as well. The addition of input chains created by GLD to the UR component's input set causes the UR-derivation system to report refutations for input sets where this was previously not possible. On the other side of this coin, extension operations against unit chains created by UR-derivation often shorten GLD refutations.

Two versions of GLD||UR have been implemented. In the first version the derivation components themselves control the distribution of the chains they create, while in the second version an extra component is used to control the distribution of chains created.

GLD||UR-1

GLD||UR-1 consists of two derivation components, one for each derivation system. Both components are responsible for examining, manipulating and transmitting to the other component, any chains that they create. Within each component, any chains created are passed to a chain control module which implements the subsumption checks, removes subsumed input chains, adds new input chains to the input set, and transmits both new chains and subsumed chains' identifiers to the other component. The flow of information is illustrated in the following figure.

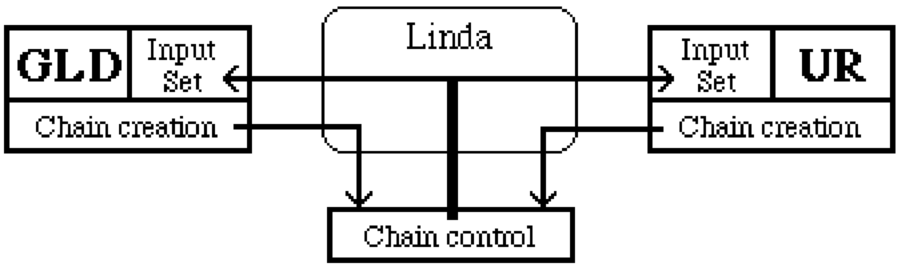


GLD||UR-1 has two problems : (i) The derivation components are responsible for examining and manipulating the chains that they create. It would be preferable for this work to be carried out external to the derivation components. (ii) The second problem could be rather more severe. As the components deal independently with the chains they create, it is possible for both to simultaneously create the same chain, determine that it should be inserted into the input set, and transmit the chain to the other component. As the receiving component accepts the new chain in good faith and adds it to its input set, both components would then have two copies of the new chain in their input sets. This would expand both search spaces.

GLD||UR-2 solves these problems.

GLD||UR-2

GLD||UR-2 consists of two derivation components and a separate chain control component. Chains created in either of the derivation components are transmitted directly to the chain control component. Here the new chains are examined for back and forward subsumption, and if necessary transmitted to the derivation components, along with identifiers for any chains that were back subsumed. The flow of information is illustrated in the following figure.



This architecture permits the two derivation components to concentrate on their derivation responsibilities. For the GLD component, this is particularly useful, as the derivation operations in GLD are fairly complex. The gain is not so important in the UR-derivation system.

5 Conclusion

GLD||UR-1 and GLD||UR-2 have been tested on a selection of reasonably hard problems. The results of testing are given in the following table. Results are given in the form <Number of derivation operations>:<Time taken>. In GLD||UR-1 and GLD||UR-2, the time is given in the column for the component that first reports a proof.

Problem	Derivation System					
	Independent		GLD||UR-1		GLD||UR-2	
	GLD	UR	GLD	UR	GLD	UR
Agatha	51 : 23	17 : 7	12 : 6	14 :	18 : 8	24 :
Group2	1413: 1158	95 : 36	21 :	95 : 12	17 :	88 : 11
School	148 : 60	No proof	134 :	48 : 52	108 : 37	50 :
Steamroller	15667 : 12218	431 : 98	154 :	431 : 104	104 : 58	347 :
Truth&Lies	2141: 2035	No proof	859 :	688 : 1203	859 :	793 : 1144

The results show the benefits of combining the two derivation systems in parallel. The relative advantage that GLD||UR has over the independent systems increases as the problems get harder. It is noteworthy that the UR component obtained a proof for two problems which it cannot prove independently.

The GLD||UR architectures allow easy addition of further components. If a new derivation component is added, it is only necessary to extend the distribution of new chain information to include the new component. In the GLD||UR-2 architecture, further lemma control components can be added to the system with no modifications to any of the existing components. As well as the addition of extra functional components, work within an individual component can be distributed. For example, the GLD search can be effectively guided by the use of a heuristic function, and the architecture of GLD makes it possible for the heuristic function to be evaluated in parallel with derivation operations.

GLD||UR is but one of a large class of parallel derivation systems, whose architecture is identified by the relative independence of the individual derivation systems that run in parallel. GLD||UR is an early development in this class of systems, and there is clearly scope for further investigation. Questions concerning appropriate combinations and numbers of components have yet to be addressed. The Prolog-Linda environment makes it possible to quickly and easily build and evaluate combinations of components. Experiments with different combinations of components are, to a large extent, unhampered by the difficulty of determining the compatibility of the individual derivation system. This is in contrast to the difficulties experienced when combining multiple refinements of resolution into a single derivation system.

References

Overbeek R., McCharen J., and Wos L., Complexity and Related Enhancements for Automated Theorem-Proving Programs, *Computers and Mathematics with Applications* 2, Pergamon Press, England, (1976), 1-16.

Shostak R.E., Refutation Graphs, *Artificial Intelligence* 7, North-Holland, Amsterdam, The Netherlands, (1976), 51-64.

Sutcliffe G., and Pinakis J., Prolog-Linda - An Embedding of Linda in muProlog, In Tsang C.P., Ed., *Proceedings of AI'90 - the 4th Australian Conference on Artificial Intelligence* (Perth, Australia, 1990), World Scientific, Singapore, (1990), 331-340.