

Some Practical Considerations

Why use ASP, what to use it for and how to debug it

Martin Brain

University of Bath

January 31, 2006

Outline

Introduction

Practical Considerations

Why use ASP?

What Characterises a Problem ASP can Solve?

What Characterises a Problem ASP can Solve Well?

Advantages of ASP

Debugging in ASP

Classification of Errors

Algorithm Based Debugging

Query based debugging

Conclusion

Introduction

Martin Brain, 2nd year PhD Student, University of Bath, UK

Thesis Topic - “The efficient usage of ASP for solving significant real world problems.”

Introduction

Practical Considerations

Why use ASP?

What Characterises a Problem ASP can Solve?

What Characterises a Problem ASP can Solve Well?

Advantages of ASP

Debugging in ASP

Classification of Errors

Algorithm Based Debugging

Query based debugging

Conclusion

Why use ASP?

- ▶ Time to solution
- ▶ Small amount of program code required
- ▶ Portability (esp. to parallel architectures)

What Characterises a Problem ASP can Solve?

- ▶ Problem and solution can be described in terms of a finite number of discrete propositions.
- ▶ Links between these are known.
- ▶ Clear idea of what you know – and what you want to find.

What Characterises a Problem ASP can Solve Well?

- ▶ Complex / 'partial' search space.
- ▶ Links are of an inferential or causal type.
- ▶ Solutions not linked to a 'topological' property of the search space.
- ▶ Can be phrased as an existential question.

Non Obvious Application Areas

- ▶ Exploring the problem / specification space
- ▶ Creating 'random' structured objects
- ▶ Verification of other algorithms

Advantages of ASP

(as versus other constraint based, declarative programming systems)

- ▶ Clear semantics
- ▶ High (native) expressive power
- ▶ Strong theoretical basis

Introduction

Practical Considerations

Why use ASP?

What Characterises a Problem ASP can Solve?

What Characterises a Problem ASP can Solve Well?

Advantages of ASP

Debugging in ASP

Classification of Errors

Algorithm Based Debugging

Query based debugging

Conclusion

Classification of Errors

- ▶ What are 'bugs'?
- ▶ What categories of bug are there?
 - ▶ Lexical : "The cat is fghjk."
 - ▶ Syntactic : "The the is cat mat on."
 - ▶ Semantic : "The cat is gaseous."
 - ▶ Conceptual : "The dog is brown."

Classification of Errors cont.

- ▶ Lexical errors: tokeniser \Rightarrow *boring*
- ▶ Syntactic errors: parser \Rightarrow *boring*
- ▶ Semantic errors: none! Every syntactically valid program is has a valid semantic definition.
- ▶ Conceptual errors: discrepancies between the program and it's specification and errors in the specification. All 'bugs' in answer set programs.

Classification of Errors cont.

The only information we have to work with is the difference between what the programmer expected and what they got.

Algorithm Based Debugging

The obvious approach. Similar to procedural programming and Prolog.

- ▶ Configurable “break points” can be added between the bounding and branching steps.
- ▶ Commands are provided for examining partial evaluations, changing branch, following implications, etc.
- ▶ Programmer can mark the blocks of the program / literal / rules of interest.
- ▶ This also allows for analysis of the efficiency of the program.
- ▶ noMoRe already implements step control similar to this.

... however ...

This approach has a few down sides.

- ▶ High cognitive load.
- ▶ Marking sections / rules takes time
- ▶ Difficult to go backwards
- ▶ Requires programmer to understand computation algorithm
- ▶ Solver specific
- ▶ A pain to implement, especially for distributed systems
- ▶ Makes no real use of locality / dependency information
- ▶ Fundamentally this is answering 'how',
the question we actually wants to ask is 'why'...

Query based debugging

- ▶ If we have answer sets and the programmer knows there are bugs in it, this is either manifested as things that are in at least one of the answer set and shouldn't be or things that are missing from at least one answer set.
- ▶ If there are no answer sets and some were expected then there are at least one literal which is expected to be in an answer set.
- ▶ Thus there are two fundamental questions:
 - ▶ Why is set S contained in answer set A ?
 - ▶ Why is set S not contained in any answer set?

Why is Set S Contained in Answer Set A

- ▶ This question is recursive.
- ▶ For a given answer set we know that an atom is in if it is supported and not if it is not supported.
- ▶ Use this to recursively justify set S .
- ▶ Create a tree of possible justifications.
- ▶ This explains the structure of the answer set but not some 'what if ...' questions, we only answer 'because ...'
- ▶ Need a catch to stop justifications with circular reasoning.

Why is Set S not Contained in any Answer Set?

- ▶ This is a harder question.
- ▶ An atom is not in any answer set if it cannot be supported or it leads to a contradiction.
- ▶ First we work 'backwards', then forwards.
- ▶ For each atom in the set and each rule that supports it, create a *situation*.
- ▶ A situation is a fragment of a partial evaluation that supports that atom with the given rule.
- ▶ If no situations can be formed then there is no way of supporting this set.

(cont.)

- ▶ Add to each situation the head of any rule that is applicable with respect to it.
- ▶ This may cause situations to be dropped due to constraints.
- ▶ Any situation left unchanged may contain circular justification. Alternatively we may need to branch to go further forwards.
- ▶ Recurse!

Open Questions

- ▶ How do we present this information to a user?
(graphs, text, expandable trees)
- ▶ How do we integrate this to produce a viable IDE?
- ▶ Can we support higher level constructs?
- ▶ How well does this scale?

Introduction

Practical Considerations

Why use ASP?

What Characterises a Problem ASP can Solve?

What Characterises a Problem ASP can Solve Well?

Advantages of ASP

Debugging in ASP

Classification of Errors

Algorithm Based Debugging

Query based debugging

Conclusion

Conclusion - Why use ASP?

- ▶ ASP is a fast way of producing a compact, portable solution
- ▶ In many cases this may be 'good enough'

Conclusion - Debugging ASP

- ▶ Most flexible approach is to use the following questions:
 - ▶ Why is set S contained in answer set A ?
 - ▶ Why is set S not contained in any answer set?
- ▶ These can answer all questions about the answer sets
- ▶ ... but there are still questions on how to return this information