Applications of Answer Set Programming

Marina De Vos Martin Brain Georg Boenn
John Fitch/ffitch

Department of Computer Science
University of Bath

19 March 2010
Datalog 2.0 - Oxford

Outline

1. TOAST
2. ANTON
3. Conclusions

Outline

1. TOAST
   - Motivation
   - The System
   - Results
2. ANTON
3. Conclusions

Motivation

- Optimisation is not Optimal!
- Superoptimisation:
  - A 'brute force' search for instruction sequences that give the correct output.

Motivation

- Optimisation is not Optimal!
- Superoptimisation:
  - A 'brute force' search for instruction sequences that give the correct output.
  - Starts with 'all possible' sequences of length 1, then 2, then...
  - First implemented by Henry Massalin, 1987
  - Used in the GCC RS/6000 / POWER port – found sequences the *chip designers* didn't think could exist!

Motivation

- Optimisation is not Optimal!
- Superoptimisation:
  - A ‘brute force’ search for instruction sequences that give the correct output.
  - Starts with ‘all possible’ sequences of length 1, then 2, then...
  - First implemented by Henry Massalin, 1987
  - Used in the GCC RS/6000 / POWER port – found sequences the *chip designers* didn’t think could exist!

Motivation

- Optimisation is not Optimal!
- Superoptimisation:
  - A ‘brute force’ search for instruction sequences that give the correct output.
  - Starts with ‘all possible’ sequences of length 1, then 2, then...
  - First implemented by Henry Massalin, 1987
  - Used in the GCC RS/6000 / POWER port – found sequences the *chip designers* didn’t think could exist!

Motivation

- Optimisation is not Optimal!
- Superoptimisation:
  - A ‘brute force’ search for instruction sequences that give the correct output.
  - Starts with ‘all possible’ sequences of length 1, then 2, then...
  - First implemented by Henry Massalin, 1987
  - Used in the GCC RS/6000 / POWER port – found sequences the *chip designers* didn’t think could exist!

Motivation

- Optimisation is not Optimal!
- Superoptimisation:
  - A ‘brute force’ search for instruction sequences that give the correct output.
  - Starts with ‘all possible’ sequences of length 1, then 2, then...
  - First implemented by Henry Massalin, 1987
  - Used in the GCC RS/6000 / POWER port
    – found sequences the *chip designers* didn’t think could exist!

A Simple Example

```
int signum (int x) {
        if (x > 0)          return 1;
        else if (x < 0)     return -1;
        else                return 0;
}
```

- Optimisers come back with sequences of 8-20 instructions including 2 to 4 branches.

Superoptimiser Output

```
! input in %i0

addcc %i0 %i0 %l1
subxcc %i0 %l1 %l2
addx %l2 %i0 %o1

! output in %o1
```

3 instructions, no branches!

Superoptimisation Tasks

Superoptimisation consists of two tasks

1. Search
2. Verification

Superoptimisation Tasks

Superoptimisation consists of two tasks

1. Search
2. Verification

TOAST

- Aim: Generating Optimal Code using Answer Set Programming
- Using ASP as the 'computational back end' for a super-optimiser.
- Requires modelling (the integer unit of) a processor.
- MIPS R2000, SPARC V7 and SPARC V8
- In this project we are interested in optimality in acyclic code and in terms of *the number of instructions used*

TOAST

- Aim: Generating Optimal Code using Answer Set Programming
- Using ASP as the 'computational back end' for a super-optimiser.
- Requires modelling (the integer unit of) a processor.
- MIPS R2000, SPARC V7 and SPARC V8
- In this project we are interested in optimality in acyclic code and in terms of *the number of instructions used*

TOAST

- Aim: Generating Optimal Code using Answer Set Programming
- Using ASP as the ‘computational back end’ for a super-optimiser.
- Requires modelling (the integer unit of) a processor.
- MIPS R2000, SPARC V7 and SPARC V8
- In this project we are interested in optimality in acyclic code and in terms of *the number of instructions used*

TOAST

- Aim: Generating Optimal Code using Answer Set Programming
- Using ASP as the 'computational back end' for a super-optimiser.
- Requires modelling (the integer unit of) a processor.
- MIPS R2000, SPARC V7 and SPARC V8
- In this project we are interested in optimality in acyclic code and in terms of *the number of instructions used*

TOAST

- Aim: Generating Optimal Code using Answer Set Programming
- Using ASP as the ‘computational back end’ for a super-optimiser.
- Requires modelling (the integer unit of) a processor.
- MIPS R2000, SPARC V7 and SPARC V8
- In this project we are interested in optimality in acyclic code and in terms of *the number of instructions used*

Flow Control

```
haveJumped(C,T) :- jump(C,T,J), jumpSize(C,J),
                   time(C,T), colour(C).

pc(C,PCV+J,T+1) :- pc(C,PCV,T), jump(C,T,J), jumpSize(C,J),
                   time(C,T), colour(C), position(C,PCV).

pc(C,PCV+1,T+1) :- pc(C,PCV,T), not haveJumped(C,T),
                   time(C,T), colour(C), position(C,PCV).

finishedAt(C,T) :- pc(C,PCV,T+1), PCV > 2,
                   time(C,T), colour(C), pcValue(C,PCV).

pc(C,1,1).
```

Instruction Definition

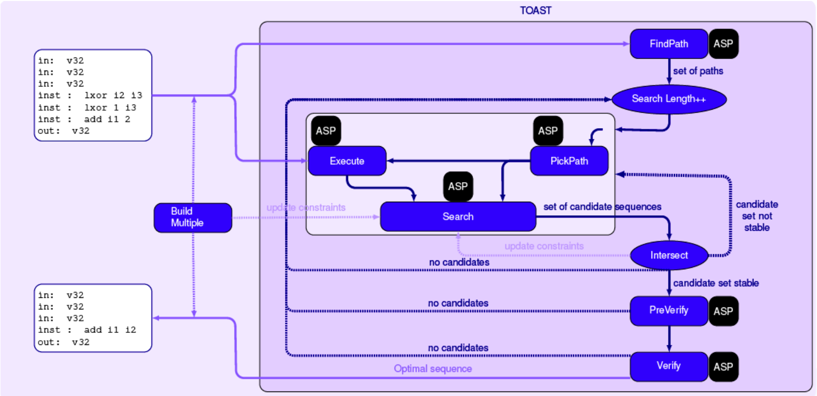
```
value(C,T,B) :- istream(C,P,lxor,R1,R2,none), pc(C,P,T),
                value(C,R1,B), -value(C,R2,B),
                register(R1), register(R2), colour(C),
                position(C,P), time(C,T), bit(B).

value(C,T,B) :- istream(C,P,lxor,R1,R2,none), pc(C,P,T),
                -value(C,R1,B), value(C,R2,B),
                register(R1), register(R2), colour(C),
                position(C,P), time(C,T), bit(B).

-value(C,T,B) :- istream(C,P,lxor,R1,R2,none), pc(C,P,T),
                 not value(C,T,B), register(R1), register(R2),
                 colour(C), position(C,P), time(C,T), bit(B).

symmetricInstruction(lxor).
```

System Diagram



Timings

	sequence4				*verifytest1*			*verifytest2*		
Solver	**ss1**	**ss2**	**ss3**	**ss4**	**8 bit**	**16 bit**	**32 bit**	**8 bit**	**16 bit**	**32 bit**
clasp-1.1.1	123.20	105.72	578.37	12355.16	0.46	0.48	15.81	0.31	0.37	8.67
smodels-2.32	123.25	266.17	6880.87	-	0.18	11.33	-	0.20	4.75	-
smodels-ie-1.0.0	123.21	281.60	1983.94	-	0.20	11.08	-	0.21	4.79	-
sup-0.2	123.22	103.46	768.36	-	0.40	3.38	-	0.15	0.14	8.70
Atoms	853	1411	2098	2941	904	2212	6940	1030	1526	2518
Rules	42740	118779	238212	410902	1622	4870	17122	3591	6591	12583

Outline

1. TOAST
2. ANTON
   - Motivation
   - The System
   - Results
3. Conclusions

Motivation

- Music is important in every culture
- Music is governed by rules
- These rules can be studied
- For the moment we study:
  - Western tonal ("Classical") music
  - "Palestrina Rules", specifically Renaissance counterpoint (Thakar 1990)

Motivation

- Music is important in every culture
- Music is governed by rules
- These rules can be studied
- For the moment we study:
  - Western tonal ("Classical") music
  - "Palestrina Rules", specifically Renaissance counterpoint (Thakar 1990)

Motivation

- Music is important in every culture
- Music is governed by rules
- These rules can be studied
- For the moment we study:
  - Western tonal ('Classical') music
  - 'Palestrina Rules', specifically Renaissance counterpoint (Thakar 1990)

Motivation

- Music is important in every culture
- Music is governed by rules
- These rules can be studied
- For the moment we study:
  - Western tonal (‘Classical’) music
  - “Palestrina Rules”, specifically Renaissance counterpoint (Thakar 1990)

Motivation

- Music is important in every culture
- Music is governed by rules
- These rules can be studied
- For the moment we study:
  - Western tonal ('Classical') music
  - "Palestrina Rules", specifically Renaissance counterpoint (Thakar 1990)

Motivation

- Music is important in every culture
- Music is governed by rules
- These rules can be studied
- For the moment we study:
  - Western tonal (‘Classical’) music
  - “Palestrina Rules”, specifically Renaissance counterpoint (Thakar 1990)

Algorithmic Composition

Predates electronic digital computers!
$\hookrightarrow$ “Mozart’s” Musikalisches Würfelspiel

improvisation	vs.	composition
automatic	vs.	assisted
harmonic	vs.	melodic
macro	vs.	micro

- Where does the next note come from?
- Composition of 4 to 16 bars
  $\hookrightarrow$ sentences and paragraphs not books

Algorithmic Composition

Predates electronic digital computers!
↪ “Mozart’s” Musikalisches Würfelspiel

improvisation	vs.	composition
automatic	vs.	assisted
harmonic	vs.	melodic
macro	vs.	micro

- Where does the next note come from?
- Composition of 4 to 16 bars
  ↪ sentences and paragraphs not books

Algorithmic Composition

Predates electronic digital computers!
$\hookrightarrow$ “Mozart’s” Musikalisches Würfelspiel

improvisation	vs.	composition
automatic	vs.	assisted
harmonic	vs.	melodic
macro	vs.	micro

- Where does the next note come from?
- Composition of 4 to 16 bars
  $\hookrightarrow$ sentences and paragraphs not books

Algorithmic Composition

Predates electronic digital computers!
$\hookrightarrow$ “Mozart’s” Musikalisches Würfelspiel

improvisation	vs.	composition
automatic	vs.	assisted
harmonic	vs.	melodic
macro	vs.	micro

- Where does the next note come from?
- Composition of 4 to 16 bars
  $\hookrightarrow$ sentences and paragraphs not books

Algorithmic Composition

Predates electronic digital computers!
$\hookrightarrow$ "Mozart's" Musikalisches Würfelspiel

improvisation	vs.	composition
automatic	vs.	assisted
harmonic	vs.	melodic
macro	vs.	micro

- Where does the next note come from?
- Composition of 4 to 16 bars
  $\hookrightarrow$ sentences and paragraphs not books

Algorithmic Composition

Predates electronic digital computers!
$\hookrightarrow$ "Mozart's" Musikalisches Würfelspiel

improvisation	vs.	composition
automatic	vs.	assisted
harmonic	vs.	melodic
macro	vs.	micro

- Where does the next note come from?
- Composition of 4 to 16 bars
  $\hookrightarrow$ sentences and paragraphs not books

Anton

- Harmonic
- Melodic
- Rhythmic

1. Automatic composition
2. Assisted composition
3. Diagnosis

⇒ with a single ASP program.

Anton

- Harmonic
- Melodic
- Rhythmic

1. Automatic composition
2. Assisted composition
3. Diagnosis

$\Rightarrow$ with a single ASP program.

Anton

- Harmonic
- Melodic
- Rhythmic

1. Automatic composition
2. Assisted composition
3. Diagnosis

$\Rightarrow$ with a single ASP program.

Anton

- Harmonic
- Melodic
- Rhythmic

1. Automatic composition
2. Assisted composition
3. Diagnosis

$\Rightarrow$ with a single ASP program.

Anton

- Harmonic
- Melodic
- Rhythmic

1. Automatic composition
2. Assisted composition
3. Diagnosis

⇒ with a single ASP program.

Anton

- Harmonic
- Melodic
- Rhythmic

1. Automatic composition
2. Assisted composition
3. Diagnosis

$\Rightarrow$ with a single ASP program.

Anton

- Harmonic
- Melodic
- Rhythmic

1. Automatic composition
2. Assisted composition
3. Diagnosis

$\Rightarrow$ with a single ASP program.

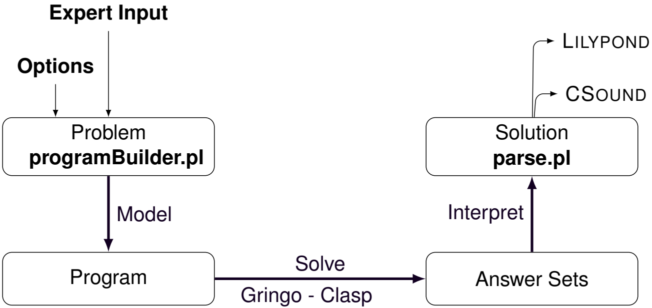
Anton

- Harmonic
- Melodic
- Rhythmic

1. Automatic composition
2. Assisted composition
3. Diagnosis

$\Rightarrow$ with a single ASP program.

System


Expert Input
Options
LILYPOND
CSOUND
Problem
programBuilder.pl
Solution
parse.pl
Model
Interpret
Program
Solve
Gringo - Clasp
Answer Sets

Rhythm

- Use of Farey sequences reduced to trees for each part
- Three layers: measures, metres, note durations
- Challenge: overlap between parts. Reduced version of Allen intervals.

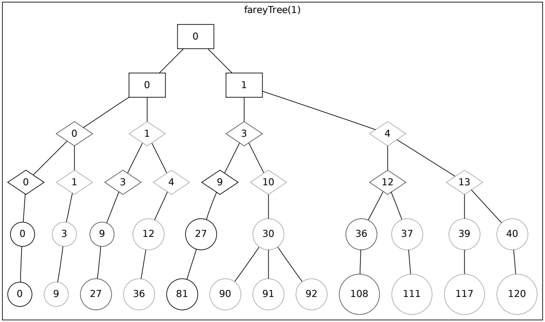
Rhythm

- Use of Farey sequences reduced to trees for each part
- Three layers: measures, metres, note durations
- Challenge: overlap between parts. Reduced version of Allen intervals.

Rhythm

- Use of Farey sequences reduced to trees for each part
- Three layers: measures, metres, note durations
- Challenge: overlap between parts. Reduced version of Allen intervals.

Rhythm Example



| 65 67 65 77 76 69 71 76 72 74 76 77
| F G F F′ E′ A B E′ C′ D′ E′ F′
| +2 −2 +12 −1 −7 +2 +5 −4 +2 +2 +1
| (((X X) (X X)) ((X (X X X)) ((X X) (X X))))

Code Fragment

```
%% Each Farey tree has a given depth
depth(F,MD + BD + DD) :- measureDepth(MD), meterDepth(F,BD),
durationDepth(F,DD).

%% the rules for the meter layer
meterLevel(F,FL) :- depth(F,DE), durationDepth(F,DD), meterDepth(F,BD),
                level(F,FL), FL <= (DE - DD), FL > (DE - (DD + BD)).

%% Map from nodes to time positions
%% Mapping increments each time a node is present
nodeStep(F,0,1).
nodeStep(F,ND,T) :- not present(F,DLL,ND), nodeStep(F,ND-1,T),
                node(F,DLL,ND), durationLeafLevel(F,DLL), ND > 0,
                partToFareyTree(P,F), partTime(P,T).
nodeStep(F,ND,T+1) :- present(F,DLL,ND), nodeStep(F,ND-1,T),
                node(F,DLL,ND), durationLeafLevel(F,DLL), ND > 0,
                partToFareyTree(P,F), partTime(P,T).

%% From this we derive a unique mapping from node to time step
timeToNode(P,1,0).
timeToNode(P,T,ND) :- present(F,DLL,ND), nodeStep(F,ND-1,T-1),
                node(F,DLL,ND), durationLeafLevel(F,DLL), ND > 0,
                partToFareyTree(P,F), partTime(P,T).
```

Music Maestro

Results

- valid pieces of music
- system to study music, not only compose or complete pieces
- 710 lines of code encoding about 48 musical rules compared to 10266 lines in Strasheela or 72345 in Bol
- Short development time
- State of the art
- but ...

Results

- valid pieces of music
- system to study music, not only compose or complete pieces
- 710 lines of code encoding about 48 musical rules compared to 10266 lines in Strasheela or 72345 in Bol
- Short development time
- State of the art
- but ...

Results

- valid pieces of music
- system to study music, not only compose or complete pieces
- 710 lines of code encoding about 48 musical rules compared to 10266 lines in Strasheela or 72345 in Bol
- Short development time
- State of the art
- but ...

Results

- valid pieces of music
- system to study music, not only compose or complete pieces
- 710 lines of code encoding about 48 musical rules compared to 10266 lines in Strasheela or 72345 in Bol
- Short development time
- State of the art
- but ...

Results

- valid pieces of music
- system to study music, not only compose or complete pieces
- 710 lines of code encoding about 48 musical rules compared to 10266 lines in Strasheela or 72345 in Bol
- Short development time
- State of the art
- but ...

Results

- valid pieces of music
- system to study music, not only compose or complete pieces
- 710 lines of code encoding about 48 musical rules compared to 10266 lines in Strasheela or 72345 in Bol
- Short development time
- State of the art
- but ...

No Rhythm

Style	Measures	Solvers		
		Clasp 1.3.2	CModels 3.79	Sup 0.4
solo	2	**0.09**	**0.20**	**0.10**
	4	**0.59**	**1.06**	**0.47**
	6	**1.42**	**2.24**	**1.22**
	8	**2.23**	**3.74**	**2.44**
	10	**4.70**	**5.68**	**3.83**
	12	**5.04**	**6.83**	**3.26**
	14	**10.85**	**8.77**	**5.79**
	16	**31.11**	**9.47**	**7.42**
duet	2	**0.23**	**0.55**	**0.27**
	4	**1.35**	**2.14**	**1.32**
	6	**6.78**	**4.69**	**5.51**
	8	21.61	**7.15**	**9.17**
	10	70.96	29.38	27.05
	12	99.53	24.90	30.58
	14	202.27	41.71	133.45
	16	443.10	115.31	81.87

Style	Measures	Solvers		
		Clasp 1.3.2	CModels 3.79	Sup 0.4
trio	2	**0.44**	**0.96**	**0.57**
	4	**1.94**	**3.04**	**2.88**
	6	**6.22**	**5.49**	**7.59**
	8	**3.98**	**9.66**	18.53
	10	116.90	**11.67**	**12.60**
	12	544.14	31.59	65.94
	14	862.75	37.65	62.88
	16	3366.75	53.63	132.54
quartet	2	**0.86**	**1.67**	**1.44**
	4	**3.57**	**4.91**	9.61
	6	128.21	**11.90**	**8.08**
	8	1081.14	24.05	75.81
	10	2775.24	81.18	127.37
	12	1777.53	201.38	75.10
	14	2803.24	348.98	259.46
	16	Time out	269.68	297.25

Rhythm

Style	Measures	Solvers		
		Clasp 1.3.2	CModels 3.79	Sup 0.4
solo	2-few	**0.69**	**0.92**	**1.00**
	2-many	**1.38**	**1.65**	**1.40**
	3-few	**1.82**	**1.92**	**3.57**
	3-many	6.38	**3.23**	6.21
	4-few	**3.65**	**4.80**	**6.85**
	4-many	37.63	8.81	9.14
	6-few	96.98	**8.49**	24.34
	6-many	1239.17	26.84	111.78
	8-few	295.64	29.51	98.76
	8-many	3374.70	51.98	379.88
duet	2-few	7.67	7.22	6.79
	2-many	14.17	20.80	20.08
	3-few	22.71	30.46	21.49
	3-many	245.12	150.91	205.07
	4-few	169.24	76.33	66.75
	4-many	2270.79	904.28	276.38
	6-few	3597.40	345.94	634.90
	6-many	Memory out	2422.77	2197.31
	8-few	Memory out	82.98	1671.19
	8-many	Memory out	Memory out	Memory out

Style	Measures	Solvers		
		Clasp 1.3.2	CModels 3.79	Sup 0.4
trio	2-few	25.18	13.42	39.00
	2-many	127.75	28.50	166.30
	3-few	284.17	49.04	264.13
	3-many	1338.05	Time out	1231.17
	4-few	1796.26	398.59	767.97
	4-many	Memory out	Time out	3310.56
	6-few	Memory out	2467.61	2890.65
	6-many	Memory out	Memory out	Memory out
quartet	2-few	325.82	45.87	91.73
	2-many	1191.95	141.64	858.02
	3-few	1019.11	204.49	295.29
	3-many	Time out	Time out	Time out
	4-few	Time out	2041.64	Time out
	4-many	Memory out	Memory out	Memory out

Outline

1. TOAST
2. ANTON
3. Conclusions

Conclusions

- State of the art application can be written in ASP
- Rapid development
- Intuitive encodings
- Good results

Conclusions

- State of the art application can be written in ASP
- Rapid development
- Intuitive encodings
- Good results

Conclusions

- State of the art application can be written in ASP
- Rapid development
- Intuitive encodings
- Good results

Conclusions

- State of the art application can be written in ASP
- Rapid development
- Intuitive encodings
- Good results

Room for Improvement

- We need faster solvers
- Grounding is a problem
  - intelligent pre-processing
  - Value and Template Variables
- Automatic optimisations - keep ASP declarative
- Need Methodologies and tools for ASP development (SEA-workshop series, Helsinki, Vienna, Bath)

Room for Improvement

- We need faster solvers
- Grounding is a problem
  - intelligent pre-processing
  - late grounding
  - Value and Template Variables
- Automatic optimisations - keep ASP declarative
- Need Methodologies and tools for ASP development (SEA-workshop series, Helsinki, Vienna, Bath)
  - Testing: regression testing, test driven development
  - Debugging: appropriate information
  - IDE: integration, support
  - Methodology: one person development to group, maintenance

Room for Improvement

- We need faster solvers
- Grounding is a problem
  - intelligent pre-processing
  - late grounding
  - Value and Template Variables
- Automatic optimisations - keep ASP declarative
- Need Methodologies and tools for ASP development (SEA-workshop series, Helsinki, Vienna, Bath)
  - Testing: regression testing, test-driven development
  - Debugging: appropriate information
  - IDE: integration, support
  - Methodology: one person development, groups, maintenance

Room for Improvement

- We need faster solvers
- Grounding is a problem
  - intelligent pre-processing
  - late grounding
  - Value and Template Variables
- Automatic optimisations - keep ASP declarative
- Need Methodologies and tools for ASP development (SEA-workshop series, Helsinki, Vienna, Bath)
  - Testing: regression testing, test-driven development
  - Debugging: appropriate information
  - IDE: integration, support
  - Methodology: one person development, groups maintenance

Room for Improvement

- We need faster solvers
- Grounding is a problem
    - intelligent pre-processing
    - late grounding
    - Value and Template Variables
- Automatic optimisations - keep ASP declarative
- Need Methodologies and tools for ASP development (SEA-workshop series, Helsinki, Vienna, Bath)
    - Debugging: appropriate information
    - IDE: integration, support

Room for Improvement

- We need faster solvers
- Grounding is a problem
  - intelligent pre-processing
  - late grounding
  - Value and Template Variables
- Automatic optimisations - keep ASP declarative
- Need Methodologies and tools for ASP development (SEA-workshop series, Helsinki, Vienna, Bath)
  - Testing: regression testing, test-driven development
  - Debugging: appropriate information
  - IDE: integration, support
  - Methodology: one person development, groups, maintenance

Room for Improvement

- We need faster solvers
- Grounding is a problem
  - intelligent pre-processing
  - late grounding
  - Value and Template Variables
- Automatic optimisations - keep ASP declarative
- Need Methodologies and tools for ASP development (SEA-workshop series, Helsinki, Vienna, Bath)
  - Testing: regression testing, test-driven development
  - Debugging: appropriate information
  - IDE: integration, support
  - Methodology: one person development, groups, maintenance

Room for Improvement

- We need faster solvers
- Grounding is a problem
  - intelligent pre-processing
  - late grounding
  - Value and Template Variables
- Automatic optimisations - keep ASP declarative
- Need Methodologies and tools for ASP development (SEA-workshop series, Helsinki, Vienna, Bath)
  - Testing: regression testing, test-driven development
  - Debugging: appropriate information
  - IDE: integration, support
  - Methodology: one person development, groups, maintenance

Room for Improvement

- We need faster solvers
- Grounding is a problem
  - intelligent pre-processing
  - late grounding
  - Value and Template Variables
- Automatic optimisations - keep ASP declarative
- Need Methodologies and tools for ASP development (SEA-workshop series, Helsinki, Vienna, Bath)
  - Testing: regression testing, test-driven development
  - Debugging: appropriate information
  - IDE: integration, support
  - Methodology: one person development, groups, maintenance

Room for Improvement

- We need faster solvers
- Grounding is a problem
  - intelligent pre-processing
  - late grounding
  - Value and Template Variables
- Automatic optimisations - keep ASP declarative
- Need Methodologies and tools for ASP development (SEA-workshop series, Helsinki, Vienna, Bath)
  - Testing: regression testing, test-driven development
  - Debugging: appropriate information
  - IDE: integration, support
  - Methodology: one person development, groups, maintenance

Room for Improvement

- We need faster solvers
- Grounding is a problem
  - intelligent pre-processing
  - late grounding
  - Value and Template Variables
- Automatic optimisations - keep ASP declarative
- Need Methodologies and tools for ASP development (SEA-workshop series, Helsinki, Vienna, Bath)
  - Testing: regression testing, test-driven development
  - Debugging: appropriate information
  - IDE: integration, support
  - Methodology: one person development, groups, maintenance