aspartame: Solving Constraint Satisfaction Problems with Answer Set Programming

M. Banbara[1] M. Gebser[2,4] K. Inoue[3] M. Ostrowski[4]
A. Peano[5] T. Schaub[4,6] T. Soh[1] N. Tamura[1] M. Weise[4]

[1]Kobe University, [2]Aalto University, [3]NII Tokyo, [4]University of Potsdam,
[5]University of Ferrara, [6]INRIA Rennes

LPNMR2015@Lexington, 30 September 2015

Motivation

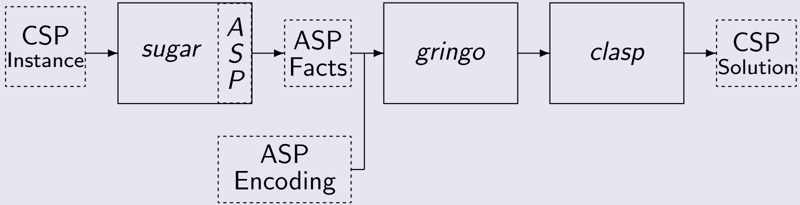
- **Answer Set Programming** (ASP)
  - General purpose approach to declarative problem solving.
  - Combination of a rich yet simple modeling language with high performance solving capacities.
- Boolean Satisfiability (SAT) showed to be an effective method to solve **Constraint Satisfaction Problems** (CSPs)
  - *sugar* encodes (finite linear) CSPs to CNF and runs modern SAT solvers.
  - *sugar* won the GLOBAL categories at the 2008 and 2009 CSP solver competitions.

Proposal

The *aspartame* framework for solving CSPs based on ASP.

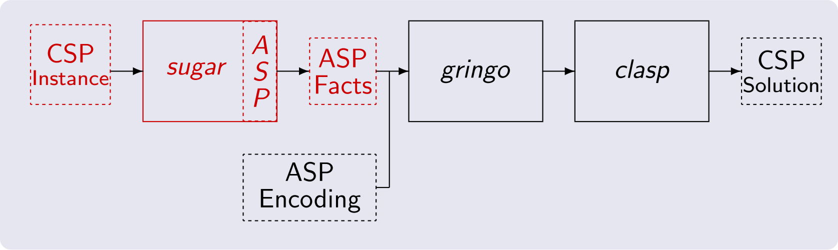
1. an ASP-based CSP Solver,
2. a library for solving CSPs as part of a logic program.

Architecture of *aspartame*



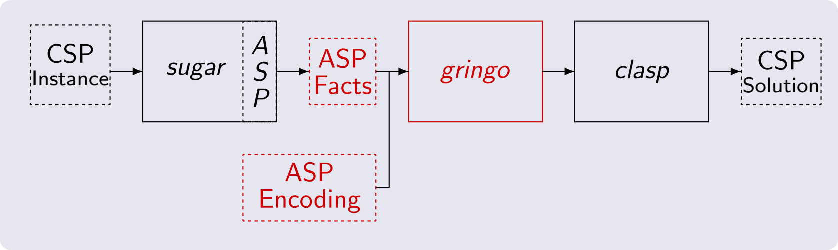
- *aspartame* accepts the XCSP (competition format) and *sugar*'s native CSP format.
- CSPs are expressed in terms of ASP facts rather than CNF.
- ASP encoding relies on **order encoding** [Tamura+.06] to map Finite-Domain to Boolean constraints.
- *aspartame* can handle **Constraint Optimization Problems**.

Architecture of *aspartame*



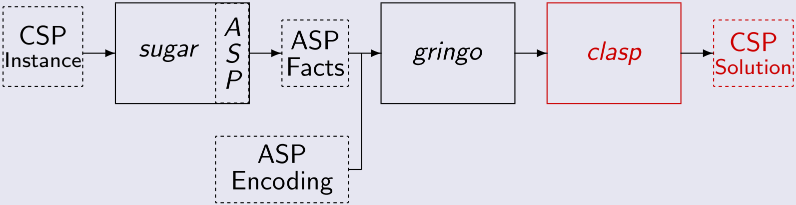
- *aspartame* accepts the XCSP (competition format) and *sugar*'s native CSP format.
- CSPs are expressed in terms of ASP facts rather than CNF.
- ASP encoding relies on **order encoding** [Tamura+.06] to map Finite-Domain to Boolean constraints.
- *aspartame* can handle **Constraint Optimization Problems**.

Architecture of *aspartame*



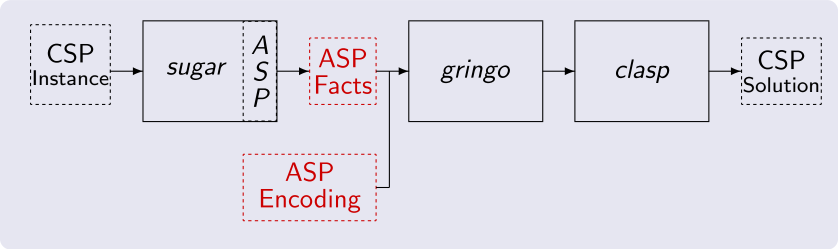
- *aspartame* accepts the XCSP (competition format) and *sugar*'s native CSP format.
- CSPs are expressed in terms of ASP facts rather than CNF.
- ASP encoding relies on **order encoding** [Tamura+.06] to map Finite-Domain to Boolean constraints.
- *aspartame* can handle **Constraint Optimization Problems**.

Architecture of *aspartame*



- *aspartame* accepts the XCSP (competition format) and *sugar*'s native CSP format.
- CSPs are expressed in terms of ASP facts rather than CNF.
- ASP encoding relies on **order encoding** [Tamura+.06] to map Finite-Domain to Boolean constraints.
- *aspartame* can handle **Constraint Optimization Problems**.

Architecture of *aspartame*



- *aspartame* accepts the XCSP (competition format) and *sugar*'s native CSP format.
- CSPs are expressed in terms of ASP facts rather than CNF.
- ASP encoding relies on **order encoding** [Tamura+.06] to map Finite-Domain to Boolean constraints.
- *aspartame* can handle **Constraint Optimization Problems**.

Summary of *aspartame* features

- **efficiency**
  aspartame is competitive to *sugar* for all instances [1] of GLOBAL categories in the 2009 CSP Competition.
- **compactness**
  ASP encoding is 700 lines long including comments/*Lua* code.
- **flexibility**
  ASP encoding is flexible enough in testing different encodings.
  - a collection of encodings for the *alldifferent* constraint.
  - Hybrid encoding of order encoding and ASP aggregates for PB constraints.

[1] except ones including extensional constraints

Finite linear CSP

Finite linear CSP

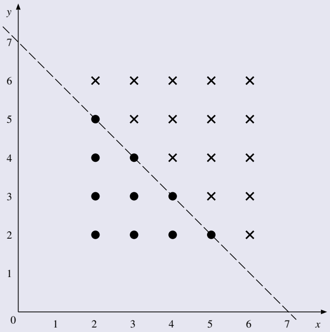
- **Integer variables** with finite domains
- **Boolean variables**
- **Arithmetic operators**: $+$, $-$, constant multiplication, etc.
- **Comparison operators**: $=$, $\neq$, $\geq$, $>$, $\leq$, $<$
- **Logical operators**: $\neg$, $\wedge$, $\vee$, $\Rightarrow$

- Many applications in AI can be formalized as CSPs.
- We can restrict the comparison to $\sum a_i x_i \leq c$ without loss of generality, where x_i's are Integer variables and a_i's and c are Integer constants.

ASP Fact Format

CSP

$$x \in \{2, 3, 4, 5, 6\}$$
$$y \in \{2, 3, 4, 5, 6\}$$
$$x + y \leq 7$$



ASP facts

```
var(int,x,(range(2,6))).
var(int,y,(range(2,6))).
constraint(1,(op(le,op(add,op(mul,1,x),op(mul,1,y)),7))).
```

Encoding: Integer variable

In order encoding, an atom p(x,a) is introduced to each Integer variable x and a value $a \in Dom(x)$.

$$\text{p}(x,a) \text{ is true} \Leftrightarrow x \leq a.$$

Integer variable $x \in \{2, 3, 4, 5, 6\}$

```
#count { p(x,2), p(x,3), p(x,4), p(x,5), p(x,6) }.
:- p(x,2), not p(x,3).   % x ≤ 2 ⇒ x ≤ 3
:- p(x,3), not p(x,4).   % x ≤ 3 ⇒ x ≤ 4
:- p(x,4), not p(x,5).   % x ≤ 4 ⇒ x ≤ 5
:- p(x,5), not p(x,6).   % x ≤ 5 ⇒ x ≤ 6
:- not p(x,6).           % x ≤ 6
```

Encoding: Integer variable (Cont.)

Possible answer sets

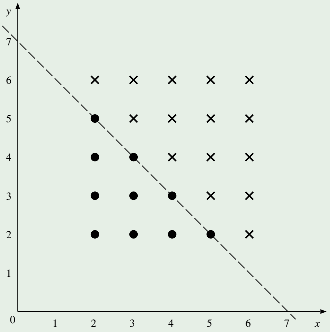
```
clasp version 3.1.2
Reading from stdin
Solving...
Answer: 1
p(x,6) % x = 6
Answer: 2
p(x,5) p(x,6) % x = 5
Answer: 3
p(x,4) p(x,5) p(x,6) % x = 4
Answer: 4
p(x,3) p(x,4) p(x,5) p(x,6) % x = 3
Answer: 5
p(x,2) p(x,3) p(x,4) p(x,5) p(x,6) % x = 2
SATISFIABLE
```

Encoding: linear comparison

Each constraint is encoded by enumerating its conflict regions instead of conflict points.

Constraint $x + y \leq 7$

1.
2.
3.
4.
5.

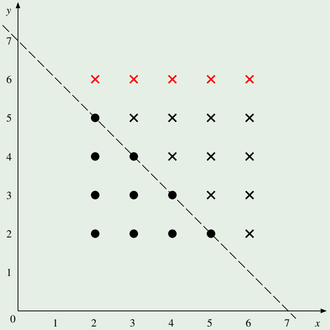


Encoding: linear comparison

Each constraint is encoded by enumerating its conflict regions instead of conflict points.

Constraint $x + y \leq 7$

1. $\neg(y \geq 6)$
2.
3.
4.
5.



Encoding: linear comparison

Each constraint is encoded by enumerating its conflict regions instead of conflict points.

Constraint $x + y \leq 7$

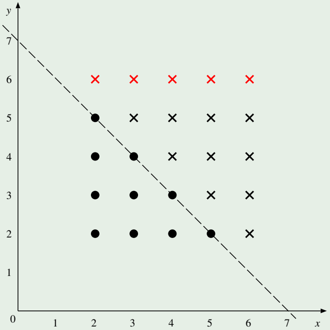
❶ `:- not p(y,5).`

❷

❸

❹

❺

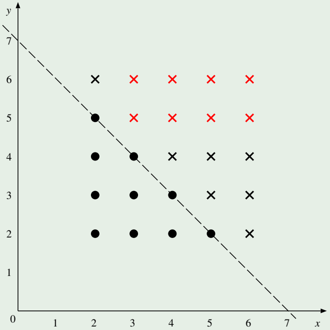


Encoding: linear comparison

Each constraint is encoded by enumerating its conflict regions instead of conflict points.

Constraint $x + y \leq 7$

1. `:- not p(y,5).`
2. $\neg(x \geq 3 \wedge y \geq 5)$
3.
4.
5.

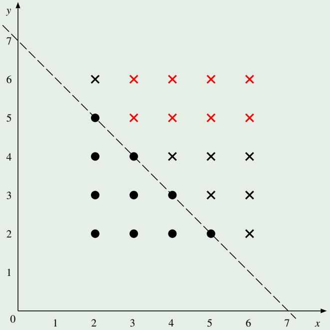


Encoding: linear comparison

Each constraint is encoded by enumerating its conflict regions instead of conflict points.

Constraint $x + y \leq 7$

1. `:- not p(y,5).`
2. `:- not p(x,2),not p(y,4).`
3.
4.
5.



Encoding: linear comparison

Each constraint is encoded by enumerating its conflict regions instead of conflict points.

Constraint $x + y \leq 7$

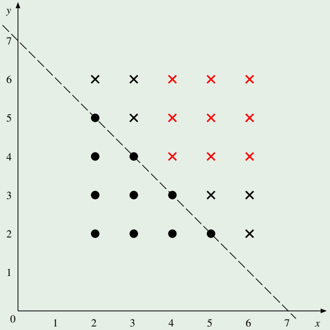
❶ `:- not p(y,5).`

❷ `:- not p(x,2),not p(y,4).`

❸ $\neg(x \geq 4 \wedge y \geq 4)$

❹

❺

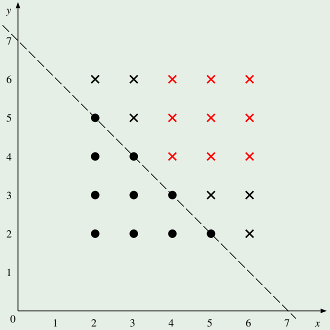


Encoding: linear comparison

Each constraint is encoded by enumerating its conflict regions instead of conflict points.

Constraint $x + y \leq 7$

1. `:- not p(y,5).`
2. `:- not p(x,2),not p(y,4).`
3. `:- not p(x,3),not p(y,3).`
4.
5.

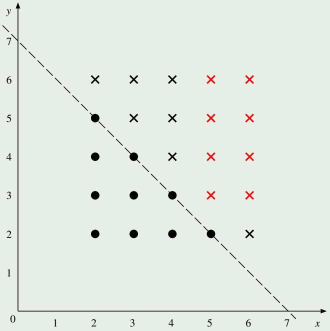


Encoding: linear comparison

Each constraint is encoded by enumerating its conflict regions instead of conflict points.

Constraint $x + y \leq 7$

1. `:- not p(y,5).`
2. `:- not p(x,2),not p(y,4).`
3. `:- not p(x,3),not p(y,3).`
4. $\neg(x \geq 5 \land y \geq 3)$
5.

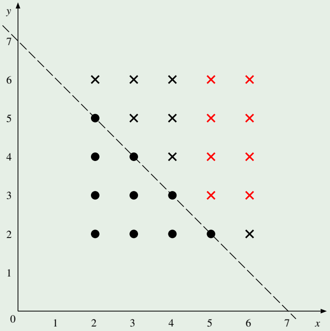


Encoding: linear comparison

Each constraint is encoded by enumerating its conflict regions instead of conflict points.

Constraint $x + y \leq 7$

1. `:- not p(y,5).`
2. `:- not p(x,2),not p(y,4).`
3. `:- not p(x,3),not p(y,3).`
4. `:- not p(x,4),not p(y,2).`
5.



Encoding: linear comparison

Each constraint is encoded by enumerating its conflict regions instead of conflict points.

Constraint $x + y \leq 7$

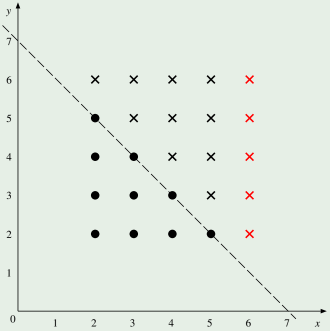
❶ `:- not p(y,5).`

❷ `:- not p(x,2),not p(y,4).`

❸ `:- not p(x,3),not p(y,3).`

❹ `:- not p(x,4),not p(y,2).`

❺ $\neg(x \geq 6)$

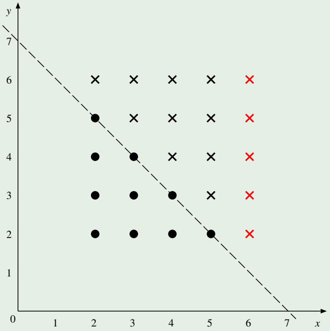


Encoding: linear comparison

Each constraint is encoded by enumerating its conflict regions instead of conflict points.

Constraint $x + y \leq 7$

1. `:- not p(y,5).`
2. `:- not p(x,2),not p(y,4).`
3. `:- not p(x,3),not p(y,3).`
4. `:- not p(x,4),not p(y,2).`
5. `:- not p(x,5).`



Encoding: linear comparison (Cont.)

```
clasp version 3.1.2
Solving...
Answer: 1
p(x,4) p(x,5) p(x,6) p(y,3) p(y,4) p(y,5) p(y,6) % (x, y) = (4, 3)
Answer: 2
p(x,5) p(x,6) p(y,2) p(y,3) p(y,4) p(y,5) p(y,6)
Answer: 3
p(x,4) p(x,5) p(x,6) p(y,2) p(y,3) p(y,4) p(y,5) p(y,6)
Answer: 4
p(x,3) p(x,4) p(x,5) p(x,6) p(y,4) p(y,5) p(y,6)
Answer: 5
p(x,2) p(x,3) p(x,4) p(x,5) p(x,6) p(y,5) p(y,6)
Answer: 6
p(x,2) p(x,3) p(x,4) p(x,5) p(x,6) p(y,4) p(y,5) p(y,6)
Answer: 7
p(x,3) p(x,4) p(x,5) p(x,6) p(y,3) p(y,4) p(y,5) p(y,6)
Answer: 8
p(x,3) p(x,4) p(x,5) p(x,6) p(y,2) p(y,3) p(y,4) p(y,5) p(y,6)
Answer: 9
p(x,2) p(x,3) p(x,4) p(x,5) p(x,6) p(y,3) p(y,4) p(y,5) p(y,6)
Answer: 10
p(x,2) p(x,3) p(x,4) p(x,5) p(x,6) p(y,2) p(y,3) p(y,4) p(y,5) p(y,6)
SATISFIABLE
```

Translation of other constraints

- Linear constraints are translated by the order encoding.
- Non-linear constraints are converted into linear forms:

Expression	Conversion
$E = F$	$(E \leq F) \land (E \geq F)$
$E \neq F$	$(E < F) \lor (E > F)$
$\max(E, F)$	x
	with $(x \geq E) \land (x \geq F) \land ((x \leq E) \lor (x \leq F))$
$\min(E, F)$	x
	with $(x \leq E) \land (x \leq F) \land ((x \geq E) \lor (x \geq F))$
$\mathrm{abs}(E)$	$\max(E, -E)$
E div c	q
	with $(E = c\,q + r) \land (0 \leq r) \land (r < c)$
E mod c	r
	with $(E = c\,q + r) \land (0 \leq r) \land (r < c)$

Translation of global constraints

- *alldifferent*$(x_1, x_2, \ldots, x_n)$ constraint is translated as follows:

$$\bigwedge_{i<j} (x_i \neq x_j) \tag{1}$$

$$\bigvee_i (x_i \geq lb + n - 1) \qquad \bigvee_i (x_i \leq ub - n + 1) \tag{2}$$

- *aspartame* encodes $x_i \neq x_j$ of (1) by either order encoding or ladder encoding [Gent+'04].
- The last two in (2) are (optional) pigeon hole clauses, and lb and ub are the lower and upper bounds of $\{x_1, x_2, \ldots, x_n\}$.
- Other global constraints (*element*, *cumulative*, etc.) are translated in a straightforward way.

Summary of *aspartame* encoding

The following shows the size of Boolean variables and clauses for encoding CSP with (maximum) domain size d.

Boolean variables for an Integer variable	$O(d)$
Clauses for an Integer variable	$O(d)$
Clauses for $\sum_{i=1}^{n} a_i x_i \le c$	$O(d^{n-1})$

- $O(d^{n-1})$ can be reduced to $O(nd^2)$ by introducing intermediate Integer variables for splitting sum expressions during preprocessing step (as *sugar* does).
- **The difference from *sugar* encoding**: The recursive structure of *aspartame* encoding allows for a similar reduction without introducing Integer variables.

Experiments

We used all instances [2] of GLOBAL categories in the 2009 CSP Competition (547 in total).

Summary of results

	#unsolved	#grounding timeouts
aspartame+split	78	24
aspartame+nosplit	75	20
aspartame+nosplit+alldiffB	72	14
aspartame (virtual best)	64	11
sugar	70	11

- *aspartame* can be competitive in solving to *sugar*.
- However, grounding is more expensive than the dedicated implementation of *sugar*.

[2] except ones including extensional constraints

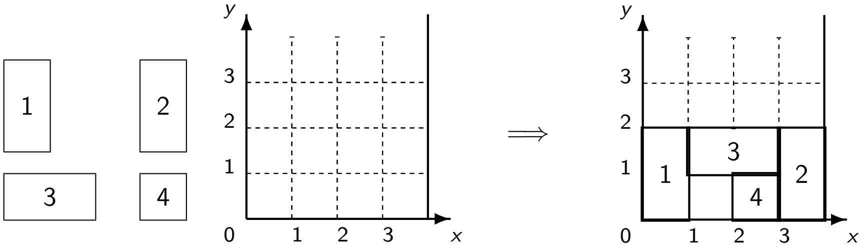
Experiments (Cont.)

	aspartame+split		*aspartame+nosplit*		*aspartame+nosplit+B*		*sugar*	
Benchmark Class	trans	solve(to^t,to)	trans	solve(to^t,to)	trans	solve(to^t,to)	trans	solve(to^t,to)
CabinetStart1(40)	53	20 (0, 0)	8	2 (0, 0)	8	2 (0, 0)	4	12 (0, 0)
QG3(7)	2	515 (0, 2)	2	515 (0, 2)	2	515 (0, 2)	2	514 (0, 2)
QG4(7)	2	278 (0, 1)	2	278 (0, 1)	2	278 (0, 1)	2	269 (0, 1)
QG5(7)	1	71 (0, 0)	1	71 (0, 0)	1	68 (0, 0)	1	60 (0, 0)
QG6(7)	4	257 (0, 1)	4	257 (0, 1)	4	257 (0, 1)	2	257 (0, 1)
QG7(7)	4	259 (0, 1)	4	259 (0, 1)	4	260 (0, 1)	2	258 (0, 1)
Allsquares(37)	162	229 (0, 4)	33	278 (0, 4)	33	281 (0, 4)	4	271 (0, 4)
AllsquaresUnsat(37)	160	683 (0, 13)	33	728 (0, 13)	32	726 (0, 13)	4	660 (0, 13)
Bibd1011(6)	30	3 (0, 0)	15	6 (0, 0)	15	6 (0, 0)	9	6 (0, 0)
Bibd1213(7)	42	20 (0, 0)	20	15 (0, 0)	20	15 (0, 0)	7	2 (0, 0)
Bibd6(10)	8	1 (0, 0)	4	1 (0, 0)	4	1 (0, 0)	3	1 (0, 0)
Bibd7(14)	9	1 (0, 0)	5	1 (0, 0)	5	1 (0, 0)	4	1 (0, 0)
Bibd8(7)	14	3 (0, 0)	7	11 (0, 0)	7	11 (0, 0)	4	2 (0, 0)
Bibd9(10)	20	2 (0, 0)	9	3 (0, 0)	9	3 (0, 0)	6	4 (0, 0)
BibdVariousK(29)	23	298 (0, 4)	14	324 (0, 5)	14	324 (0, 5)	6	266 (0, 3)
bqwh15106_glb(10)	1	0 (0, 0)	1	0 (0, 0)	0	0 (0, 0)	1	0 (0, 0)
bqwh18141_glb(10)	1	0 (0, 0)	1	0 (0, 0)	0	0 (0, 0)	1	0 (0, 0)
Cjss(10)	97	1085 (0, 6)	86	1091 (0, 6)	87	1091 (0, 6)	24	944 (0, 5)
Compet02(20)	1165	200 (2, 2)	1164	200 (2, 2)	842	19 (0, 0)	22	9 (0, 0)
Compet08(16)	90	16 (0, 0)	91	16 (0, 0)	233	455 (0, 4)	73	463 (0, 4)
CostasArray(11)	15	381 (0, 2)	16	514 (0, 3)	6	343 (0, 2)	2	362 (0, 2)
LatinSquare(10)	2	180 (0, 1)	2	180 (0, 1)	0	180 (0, 1)	1	180 (0, 1)
MagicSquare(18)	1208	1103 (11, 11)	1444	1400 (14, 14)	1442	1401 (14, 14)	629	756 (6, 7)
Medium(5)	1717	1446 (4, 4)	1721	1446 (4, 4)	1507	34 (0, 0)	31	10 (0, 0)
Nengfa(3)	777	5 (0, 0)	770	5 (0, 0)	9	1 (0, 0)	4	6 (0, 0)
pigeons_glb(19)	0	0 (0, 0)	0	0 (0, 0)	0	0 (0, 0)	1	0 (0, 0)
PseudoGLB(100)	142	482 (7, 26)	12	382 (0, 18)	12	382 (0, 18)	95	488 (5, 26)
Rcpsp(39)	1	0 (0, 0)	0	0 (0, 0)	0	0 (0, 0)	1	0 (0, 0)
RcpspTighter(39)	1	0 (0, 0)	0	0 (0, 0)	0	0 (0, 0)	1	0 (0, 0)
Small(5)	87	1 (0, 0)	87	1 (0, 0)	62	1 (0, 0)	4	1 (0, 0)
Total(547)	163	273 (24, 78)	124	274 (20, 75)	110	264 (14, 72)	44	252 (11, 70)

aspartame as a library for solving CSPs

Two Dimensional Strip Packing (2sp) Problem

For given a set of rectangles and one large rectangle (called strip), the goal of 2sp problem is to find the minimum strip height such that all rectangles are packed into the strip without overlapping.



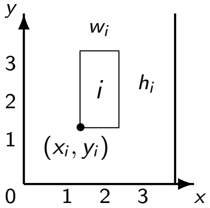
Input rectangles   Strip (width=4)   Optimum solution

ASP facts

```
#const lb = 0.  #const ub = 4.
width(4). r(1,1,2). r(2,1,2). r(3,2,1). r(4,1,1).
```

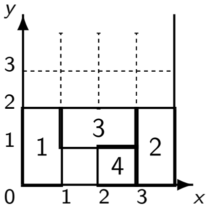
Direct Constraint Modeling of 2sp

Most direct modeling would be introducing a pair of integer variables (x_i, y_i) that represents the position of lower left coordinates of each rectangle i.



Strip (width=4)

$\Longrightarrow$



Optimum solution

$$(x_1, y_1) = (0, 0)$$
$$(x_2, y_2) = (3, 0)$$
$$(x_3, y_3) = (1, 1)$$
$$(x_4, y_4) = (2, 0)$$

And then, we enforce non-overlapping constraints for every two different rectangles i and j $(i < j)$.

$$(x_i + w_i \le x_j) \vee (x_j + w_j \le x_i) \vee (y_i + h_i \le y_j) \vee (y_j + h_j \le y_i)$$

Encoding of 2sp

2sp.clp

```
var(int, x(I), range(0, W-X)) :- r(I,X,Y), width(W).
var(int, y(I), range(0,ub-Y)) :- r(I,X,Y).
var(int, height, range(lb,ub)).
objective(minimize, height).

1 { le(x(I),XI,x(J)) ;
    le(x(J),XJ,x(I)) ;
    le(y(I),YI,y(J)) ;
    le(y(J),YJ,y(I)) } :- r(I,XI,YI), r(J,XJ,YJ), I < J.
le(y(I),Y,height) :- r(I,X,Y).

wsum(op(le,op(add,op(mul,1,X),op(mul,-1,Y)),-C)) :- le(X,C,Y).
```

- `le(x,c,y)` is used to express $x + c \leq y$
- The blue-colored predicates are defined in *aspartame* encoding.
- This encoding can be highly competitive in performance to a SAT-based approach for solving 2sp [Soh+'10].

Conclusion

We presented an alternative approach to solving finite linear CSPs based on ASP.

aspartame

`http://www.cs.uni-potsdam.de/aspartame/`

Future Work

- Improvement of the current experimental support for linear CSPs in *gringo*
- Investigation of alternative encodings
- Investigation of Multi-shot CSP solving based on ASP
- Investigation of search heuristics of CSP with *hclasp* features