Lecture 5: Modelling Aspects

Outline

1. An application: product configuration
2. Principles for relation design

1. AN APPLICATION: PRODUCT CONFIGURATION

- *Product configuration* has been a research topic in the area of artificial intelligence (AI) since 1980s.
- At present, it is already a commercially successful application of AI and ASP in particular, see e.g. `http://www.variantum.com/`.
- Product configuration domain exhibits dynamic aspects which are difficult to model using ordinary constraint satisfaction (CSP).
- The rule types of the `smodels` system were developed in close cooperation with experts from the product configuration domain.
- As demonstrated below, they lend themselves for representing knowledge that is typically involved in *configuration models*.

Configuration Models

- Typical configuration models combine a number of *requirements*, *conditional choices*, and *constraints* with minimality.
- Given these features of configuration modelling, it is nontrivial to define which sets of components represent *valid configurations*.
- The situation becomes more complex if a form of optimization (production costs, prices, capacities, . . .) is necessary.
- The basic functionalities of a *product configurator* include:
  1. *checking* whether a configuration is valid with respect to a configuration model, and
  2. *generating* one or all valid configurations for a configuration model augmented by a set of additional requirements.

Example

Consider the problem of configuring a *personal computer* (PC):

1. PC components include various kinds of CPUs, hard disks, CD/DVD ROMs, display controllers, keyboards, connectors, etc.
2. Typically, at least one mass storage unit is required for a PC: either an IDE hard disk, SCSI hard disk, or an external USB disk.
3. The layout of the keyboard must be selected.
4. Optionally/by default, a CD/DVD ROM drive could be included.
5. The choice of a SCSI disk implies a SCSI controller for the PC.

Forms of Configuration Knowledge

- A typical configuration model represents a number of choices for components to be included in a valid configuration.
- Choices may depend dynamically on each other.
- Examples of other relevant pieces of knowledge:
  1. A set of elements *requires* the presence of some other element.
  2. A set of elements is mutually *incompatible*.
  3. An element is *optional*.
  4. An element is included by *default*.

Configuration Rule Language

A number of rule types are useful for representing configuration knowledge to form rule-based models of configurable products:

$a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m.$	Requirements (R_r)
$a_1 \mid \ldots \mid a_h \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m.$	Choices (R_c)
$a_1 \oplus \ldots \oplus a_h \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m.$	Exclusive choices (R_e)
$\leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m.$	Incompatibilities (R_i)

- A *configuration model* R is a union $R_r \cup R_c \cup R_e \cup R_i$ of rules.
- A shorthand $B, \sim C$ is introduced for rule bodies

$$b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m.$$

Semantics in Brief

Definition. A set S of *components* satisfies a rule body $B, \sim C$ iff $B \subseteq S$ and $C \cap S = \emptyset$. The satisfaction of heads is summarized below.

1. $S \models a \iff a \in S.$
2. $S \models a_1 \mid \ldots \mid a_h \iff \{a_1, \ldots, a_h\} \cap S \neq \emptyset.$
3. $S \models a_1 \oplus \ldots \oplus a_h \iff |\{a_1, \ldots, a_h\} \cap S| = 1.$
4. $S \not\models \bot.$

Definition. The set R^S of reduced rules contains $a \leftarrow B$ iff a appears in the head of the respective rule, $S \models a$, and $S \models \sim C$.

Definition. A configuration S is R-valid iff $S = \mathrm{LM}(R^S)$ and $S \models R$.

Remark. The last requirement for R-validity enforces the satisfaction of configuration rules in R!

Example

Consider a set of rules R for configuring computer hardware:

Computer.

IDEdisk | SCSIdisk | Floppy $\leftarrow$ Computer.

FinKB $\oplus$ EngKB $\leftarrow$ Computer.

SCSIcontroller $\leftarrow$ SCSIdisk.

- Test the $R \cup \{\text{FinKB.}\ \}$-validity of the sets below:

$S_1 = \{\text{Computer}, \text{SCSIdisk}\}.$

$S_2 = \{\text{Computer}, \text{IDEdisk}, \text{FinKB}, \text{SCSIcontroller}\}.$

$S_3 = \{\text{Computer}, \text{SCSIdisk}, \text{FinKB}, \text{SCSIcontroller}\}.$

- Determine $\mathrm{LM}((R \cup \{\text{FinKB.}\ \})^{S_3})$ and verify $S_3 \models R \cup \{\text{FinKB.}\ \}$.

Translation into ASP

- Configuration rules for requirements and incompatibilities can be directly viewed as normal rules and constraints.
- (Exclusive) choices can be expressed using choice rules having lower and/or upper bounds:

$$a_1 \mid \ldots \mid a_h \leftarrow B, \sim C. \quad \rightsquigarrow \quad 1\,\{a_1, \ldots, a_h\} \leftarrow B, \sim C.$$
$$a_1 \oplus \ldots \oplus a_h \leftarrow B, \sim C. \quad \rightsquigarrow \quad 1\,\{a_1, \ldots, a_h\}\,1 \leftarrow B, \sim C.$$

- Minimize/maximize statements capture optimization criteria.
- Let $\mathrm{Tr}(R)$ denote the respective translation of a configuration model R where bounds have been removed from the heads of rules.

Theorem. A set of components S is R-valid iff $\mathrm{Tr}(S) \in \mathrm{SM}(\mathrm{Tr}(R))$.

2. PRINCIPLES FOR RELATION DESIGN

- The semantics of answer set programs that involve variables and relation symbols is defined in terms of Herbrand interpretations:

$$\forall M \subseteq \mathrm{Hb}(P)\colon\ M \in \mathrm{SM}(P) \text{ iff } M = \mathrm{LM}(\mathrm{Gnd}(P)^M).$$

- Given a stable model $M \subseteq \mathrm{Hb}(P)$ and a relation symbol R of arity n, one can recover the interpretation of R over $\mathrm{Hu}(P)$ by setting

$$R^M = \{\langle t_1, \ldots, t_n \rangle \mid R(t_1, \ldots, t_n) \in M\}.$$

- Thus any logic program can be viewed as a *definition* of a set of relations—whose design deserves a good deal of attention as such.
- Also, principles from relational database design can be applied while keeping in mind the relationship between SQL and rules.

Domain Specifications

- It is good to know/estimate the cardinalities of the domains of the variables involved in a program.
- Such an analysis provides a basis for estimating the size of $\mathrm{Gnd}(P)$—or the number of instances of individual rules.

Example. Recall a snapshot from our SuDoku program:

$$\mathsf{Number}(1). \quad \ldots \ \mathsf{Number}(9).$$
$$\mathsf{Border}(1). \quad \mathsf{Border}(4). \quad \mathsf{Border}(7).$$
$$\mathsf{Region}(X,Y) \leftarrow \mathsf{Border}(X), \mathsf{Border}(Y).$$

For the least model M of the respective ground program:

$$|\mathsf{Number}^M| = 9,\ |\mathsf{Border}^M| = 3, \text{ and } |\mathsf{Region}^M| = 3 \times 3 = 9.$$

Complexity of Individual Rules

- Each rule of a logic program is a part of the definition(s) of relation symbol(s) mentioned in its head.
- Given the domains of *global* variables $x_1, \ldots, x_n$ that appear in a rule r, the rule can be viewed as a relation $\mathrm{Gnd}(r)$ over $\mathrm{Hu}(P)$:

$$\langle t_1, \ldots, t_n \rangle \in \mathrm{Gnd}(r) \text{ iff } r(t_1, \ldots, t_n) \in \mathrm{Gnd}(P)$$

where $r(t_1, \ldots, t_n) = r\{x_1/t_1, \ldots, x_n/t_n\}$.

- Recall that $\mathrm{Gnd}(r) = \mathrm{Hu}(P)^n$ by the definition of $\mathrm{Gnd}(P)$ but intelligent grounders try to generate far fewer instances of r.
- Such a sound optimization activity relies on the knowledge about the domains of variables $x_1, \ldots, x_n$ involved in a rule.

Example

To this end, let us analyze rules from the SuDoku program on the basis of the domain sizes that were just pointed out.

- Since $|\text{Number}| = 9$, we will get $9^2 = 81$ instances of the constraint

$$\leftarrow 2\,\{\text{Value}(x,y,n) \mid \text{Number}(n)\}, \text{Number}(x;y).$$

- Note that n above is a *local* variable that will increase the number of conditions in the cardinality constraint up to $|\text{Number}| = 9$.
- The number of instances is $|\text{Number}| \times |\text{Region}| = 9^2 = 81$ for

$$1\,\{\text{Value}(x,y,n) \mid \text{Number}(x;y), x1 \leq x \leq x1+2,$$
$$y1 \leq y \leq y1+2\}\,1 \leftarrow \text{Number}(n), \text{Region}(x1,y1).$$

- Each choice involves $3^2 = 9$ instances of the Value predicate.

Splitting Relations

- Suppose that the first $k < n$ arguments of an n-ary relation symbol R provide a key for the tuples involved in the respective relation.
- Such a relation can be split into $n-k$ relations of arity $k+1$:

$$\langle t_1, \ldots, t_n \rangle \in R^M \text{ iff}$$
$$\langle t_1, \ldots, t_k, t_{k+1} \rangle \in R_1^M \text{ and } \ldots \text{ and } \langle t_1, \ldots, t_k, t_n \rangle \in R_{n-k}^M.$$

- The relation symbols $R_1, \ldots, R_{n-k}$ have less arguments and save space if the introduction of unnecessary variables is avoided.
- It is possible to recover R in terms of a rule

$$R(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n) \leftarrow$$
$$R_1(x_1, \ldots, x_k, x_{k+1}), \ldots, R_{n-k}(x_1, \ldots, x_k, x_n).$$

but this may be impractical due to the size of the ground program.

Example

Consider a less optimal formulation of the 8-queens problem:

$$\text{Number}(1;2;3;4;5;6;7;8).$$
$$1\,\{\text{Cell}(q,x,y) \mid \text{Number}(x;y)\}\,1 \leftarrow \text{Number}(q).$$
$$\leftarrow \text{Cell}(q1,x,y1;\ q2,x,y2), q1 \neq q2, y1 \neq y2, \text{Number}(q1;q2;x;y1;y2).$$

- Since $|\text{Number}| = 8$, the choice rule has 8 instances—each involving 64 cells. The constraint has $56^2 \times 8 = 25088$ instances!
- It is possible to split $\text{Cell}(q,x,y)$ into $\text{Column}(q,x)$ and $\text{Row}(q,y)$.

$$1\,\{\text{Column}(q,x) \mid \text{Number}(x)\}\,1 \leftarrow \text{Number}(q).$$
$$\leftarrow \text{Column}(q1,x;\ q2,x), q1 \neq q2, \text{Number}(q1;q2;x).$$

☞ The number of constraints drops down to $56 \times 8 = 448$.

Symmetries

- Many problems that have been addressed using ASP techniques are subject to *combinatorial explosion*: the number of cases to consider grows as problem-specific parameters grow.
- Symmetries decrease the efficiency of ASP in several ways.
  1. Individual relations may reserve extra space due to symmetries.
  2. When computing all/several answer sets, symmetric copies of some or all answer sets are encountered multiple times.
  3. Symmetric candidates for answer sets, which turn out not to be answer sets, are excluded repeatedly during the search.
- Many sources of symmetry can be avoided by careful design.

Symmetric Relations

- Many binary relations are symmetric by nature.
- It is possible to halve the space reserved by such relations by enforcing asymmetry in terms of additional constraints.

Example. Matches organized in a sports tournament are symmetric (the fact that team x plays team y means that team y plays team x):

$$\mathsf{Team}(1). \quad \ldots\ \mathsf{Team}(12).$$
$$\mathsf{Match}(x,y) \leftarrow \mathsf{Team}(x),\ \mathsf{Team}(y),\ x \neq y.$$

1. Now $|\mathsf{Team}| = 12$ and $|\mathsf{Match}| = |\mathsf{Team}|^2 - |\mathsf{Team}| = 132$.
2. This number can be halved to 66 by substituting $x < y$ for $x \neq y$.
3. Then the asymmetry of Match must be taken into account.

Symmetric Answer Sets

Example. Let us reconsider the formulation of the 8-queens problem:

$\mathsf{Number}(1;2;3;4;5;6;7;8).$

$1\,\{\mathsf{Column}(q,x) \mid \mathsf{Number}(x)\}\,1 \leftarrow \mathsf{Number}(q).$

$\leftarrow \mathsf{Column}(q1,x;\ q2,x),\ q1 \neq q2,\ \mathsf{Number}(q1;q2;x).$

$1\,\{\mathsf{Row}(q,x) \mid \mathsf{Number}(x)\}\,1 \leftarrow \mathsf{Number}(q).$

$\leftarrow \mathsf{Row}(q1,x;\ q2,x),\ q1 \neq q2,\ \mathsf{Number}(q1;q2;x).$

$\mathsf{DC}(q1,q2,|x1-x2|) \leftarrow \mathsf{Column}(q1,x1;q2,x2),\ \mathsf{Number}(q1;x1;q2;x2).$

$\mathsf{DR}(q1,q2,|y1-y2|) \leftarrow \mathsf{Row}(q1,y1;q2,y2),\ \mathsf{Number}(q1;y1;q2;y2).$

$\leftarrow \mathsf{DC}(q1,q2,d),\ \mathsf{DR}(q1,q2,d),\ \mathsf{Number}(q1;q2;d).$

☞ Due to identities of the queens, the number of answer sets gets multiplied by $8! = 40320$ and it becomes as high as $3709440 = 8! \times 92$.

Reducing Symmetries

The factor of 8! can be avoided altogether if the identities of queens are not represented and only cells are reserved for them.

$\mathsf{Number}(1;2;3;4;5;6;7;8).$

$8\,\{\mathsf{Queen}(x,y) \mid \mathsf{Number}(x;y)\}\,8.$

$\leftarrow \mathsf{Queen}(x,y1;x,y2),\ y1 \neq y2,\ \mathsf{Number}(x;y1;y2).$

$\leftarrow \mathsf{Queen}(x1,y;x2,y),\ x1 \neq x2,\ \mathsf{Number}(x1;x2;y).$

$\leftarrow \mathsf{Queen}(x1,y1;x2,y2),\ x1 \neq x2,\ y1 \neq y2,\ |x1-x2| = |y1-y2|,$
$\quad \mathsf{Number}(x1;y1;x2;y2).$

- The number of answer sets for this program is 92.
- Certain symmetries still persist (consider rotation and reflection).

Exploiting Default Negation

- Due to minimality, one can concentrate on specifying which things are true in a model M—others are *false by default*.
- Phrased in terms of an n-ary relation symbol R: we aim to state which tuples $\langle t_1,\ldots,t_n\rangle$ are in R^M—others are *out by default*.
- This line of reasoning works fine for relatively "small" relations but may create unnecessarily large relations otherwise.
- The question is which one is bigger: R^M or its complement?
  1. The smaller one can be used for knowledge representation.
  2. The complement is available through default negation $\sim$.
- One might ask an analogous question at the level of stable models!

Example

Consider the following definitions of equality and difference:

$$\mathsf{Number}(1).\quad \ldots\ \mathsf{Number}(n).$$
$$\mathsf{Equal}(x,x) \leftarrow \mathsf{Number}(x).$$
$$\mathsf{Differ}(x,y) \leftarrow \sim\!\mathsf{Equal}(x,y), \mathsf{Number}(x;y).$$

(A new relation symbol is introduced for the complement!)

➤ The size of the domain $|\mathsf{Number}| = n$ is parameterized and could be specified separately, e.g., from the command line of `lparse`.

➤ The cardinalities $|\mathsf{Equal}|$ and $|\mathsf{Differ}|$ are n and $n^2 - n$, respectively, which suggests that the former is preferably represented.

OBJECTIVES

➤ You are aware/can name one commercial application area of ASP.

➤ You know the main features of the product configuration domain and are able to express them using choice rules and constraints.

➤ You are familiar with a number of design principles that can be used to cut down the size of the resulting ground program.

➤ You are able to calculate/estimate the sizes of relations involved in your own programs and make design decisions in this respect.

TIME TO PONDER

Consider the following program for the tournament scheduling problem:

```
team(1..n).  week(1..n-1).  field(1..n/2).

1 { schedule(W,F,T1,T2):team(T1):team(T2):T1<T2 } 1 :-
  week(W), field(F).

:- 2 { schedule(W,F,T1,T2):week(W):field(F) },
   team(T1), team(T2), T1<T2.

:- 2 { schedule(W,F1,T,T1):field(F1):team(T1):T<T1,
       schedule(W,F2,T2,T):field(F2):team(T2):T2<T },
   team(T), week(W).
```

Are there any symmetries once the `fields`-predicate is removed?