Comparing Different Logic-Based Representations of Automotive Parts Lists

Carsten Sinz
JKU Linz
Austria

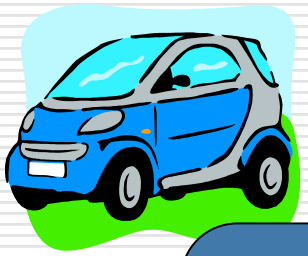
Overview

- Background: batch configuration in automotive industry (DaimlerChrysler)
- Order processing in two stages:
  1. Configuration
  2. Parts list generation

- Challenge: efficient transformation from orders to parts lists needed
  - Compact, intelligible, easily maintainable
- Comparison of 5 such transformations

Problem Description

Motor 1,
Color 20,
Gear shift 2



Motor 2,
Color 21,
Gear shift 3

How to construct mapping from configuration domain to parts list?

P1, P3, P5, ...

P2, P3, P4, ...

Configuration and Parts List Generation

- Configuration and parts list generation typically separated
- Configuration:
  - Based on properties, functionalities, features
  - Checks feasibility
- Parts list determined by mapping from orders to parts
  - No configuration on parts level

Order Processing at DaimlerChrysler

- Order processing in two stages:
  1. Order completion & consistency check
  2. Parts list generation
- Order = Set of Boolean variables (codes)
  - Codes represent equipment options
  - E.g.: { M1, G2, L, C1, ...,X, Y, ... }
- Steps 1 and 2 controlled by propositional rules that are stored in database tables
  - Examples: $G2 \wedge \neg X \Rightarrow \neg M2$,

    $if(M1 \wedge Y \wedge \neg G1)\ select(part10)$

Formal Problem Statement

- Given set $\mathcal{C}$ of codes and set $\mathcal{P}$ of parts
- How can a parts list mapping M: P($\mathcal{C}$) → P($\mathcal{P}$) be represented?
- Comments:
  - Parts list typically decomposed into modules (one for each assembly position, e.g. mirror of left front door):
    $M(o) = M_1(o) \cup M_2(o) \cup ... \cup M_k(o)$ with
    M_i: P($\mathcal{C}$) → P($\mathcal{P}_i$) for k assembly positions
  - Mapping for each module typically functional, i.e. M_i: P($\mathcal{C}$) → $\mathcal{P}_i$, thus $M(o) = \{ M_i(o) \mid 1 \le i \le k \}$
    (parts sets $\mathcal{P}_i$ possibly extended by null-part)

Solution 1: Variant Table

- Explicit part assignment for all possible (valid) configurations
- Example:
  - 3 codes (A,B,C)
  - 4 parts (P1,...,P4)
  - valid configurations:
    - at least one of A, B, C
    - A and B implies C
- Problem: Table rapidly grows too large

variant	*A*	*B*	*C*	part
1	X			P1
2		X		P2
3			X	–
4	X		X	P3
5		X	X	P2
6	X	X	X	P4

Solution 1 Feasible?

- Largest part map for Mercedes E-Class:
  - 135 codes, 27 different parts at one assembly position
  - Table: would contain (up to) $2^{135} \approx 4.36 \times 10^{40}$ lines
- Consequence: Explicit representation of variant table infeasible!

Solution 2: Propositional Encoding

- Implicit (symbolic) representation of sets of code-lists
- Pro:
  - More compact
  - Enables use of efficient data structures like BDDs to represent formulas
- Con: may still be hard to maintain

variant	A	B	C	part
1	X			P1
2		X		P2
3			X	–
4	X		X	P3
5		X	X	P2
6	X	X	X	P4

parts rule	part
$A \wedge \neg B \wedge \neg C$	P1
$\neg A \wedge B$	P2
$A \wedge \neg B \wedge C$	P3
$A \wedge B \wedge C$	P4

Solution 3a: Propositional Encoding with Implicit Negation

- Build term (conjunction of literals) for each row of variant table, then strip off negated variables
- Pro:
  - Compact in some cases
  - Relatively easy to maintain
- Con:
  - Evaluation of formulas requires pre-processing step
  - Size: one term for each row of variant table
  - Absorption law not valid

variant	A	B	C	part
1	X			P1
2		X		P2
3			X	–
4	X		X	P3
5		X	X	P2
6	X	X	X	P4

IN parts rule	part
A	P1
$B \vee (B \wedge C)$	P2
$A \wedge C$	P3
$A \wedge B \wedge C$	P4

Solution 3b: Motivation

- „Bird“ example from knowledge representation:

$$bird(x) \Rightarrow flies(x)$$

$$bird(x) \wedge Penguin(x) \Rightarrow \neg\ flies(x)$$

$$bird(x) \wedge Penguin(x) \wedge onPlane(x) \Rightarrow flies(x)$$

- Problem: rule overlaps
- Solutions:
  a) Take most specific rule
  b) Avoid rule overlaps

Solution 3b: Propositional Encoding with Implicit Exclusion

- Idea: implicitly exclude overlaps of formulas
- Realization: Conjunctively add negated terms of other parts' formulas (if not less specific)
- Pro:
  - Relatively compact
  - Avoids explicit negations
- Con:
  - Relatively hard to understand
  - Requires additional decoding step of formulas

variant	A	B	C	part
1	X			P1
2		X		P2
3			X	–
4	X		X	P3
5		X	X	P2
6	X	X	X	P4

IE parts rule	part
A	P1
$B \vee (B \wedge C)$	P2
$A \wedge C$	P3
$A \wedge B \wedge C$	P4

Solution 4a: Propositional Encoding with Rule Priority

- Assign evaluation priority to rules; check formulas in order of decreasing priority
- Pro:
  - Compact representation
  - Avoids overlaps
- Con:
  - Some rules may never fire (if more specific than rule with higher priority)
  - May become hard to maintain if many rules are involved

variant	A	B	C	part
1	X			P1
2		X		P2
3			X	–
4	X		X	P3
5		X	X	P2
6	X	X	X	P4

RP parts rule	priority	part
$A \wedge B \wedge C$	3	P4
B	2	P2
$A \wedge C$	2	P3
A	1	P1

Solution 4b: Cascaded Conditions

Maintenance example:
Assume that { A, B } becomes constructible, but should not belect a part.
Which changes are needed?

variant	A	B	C	part
1	X			P1
2		X		P2
3			X	–
4	X		X	P3
5		X	X	P2
6	X	X	X	P4

if $A \wedge B \wedge C$ **then** select(P4)
else if $A \wedge C$ **then** select(P3)
else if B **then** select(P2)
else if A **then** select(P1)

Industrial Use

- SAP Automotive:
  - Rule priority
- DaimlerChrysler Mercedes cars:
  - Prop. encoding with implicit exclusion

Conclusion

- Parts list mappings are simple from a theoretical point of view
- Finding the right formalism (apprehensible, easily maintainable) is not straightforward
- Rule compilation should be considered a programming task
  - Apply software engineering methods: style-guides, debugging, testing, verification, ...
  - Tool support needed