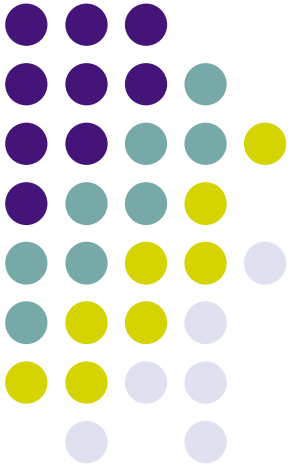
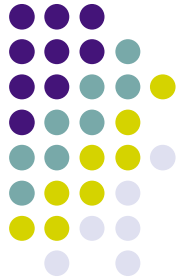
Knowledge Compilation for Product Configuration

Carsten Sinz
Symbolic Computation Group
Wilhelm-Schickard-Institut for Computer Science
University of Tübingen, Germany

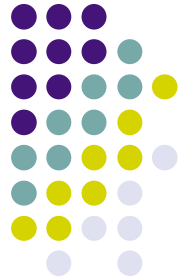


Starting Point



- Usually large series of configuration runs for same *domain description* resp. *knowledge base*
- Generation of each single consistent configuration can be computationally hard
- Only “small“ differences between different configuration runs
- Time required to solve a configuration instance hardly predictable

Knowledge Compilation: Introduction

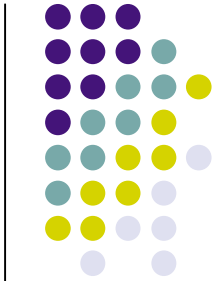


- General Idea:

  Pre-compute (compile) some or all logical consequences of the knowledge base

- Consequences:
  - Solving individual problem instances may be accelerated
  - Overall run-time may be reduced
  - Time required to solve a problem instance may become predictable (constant, at best)

Knowledge Compilation: Example



Theory DD:

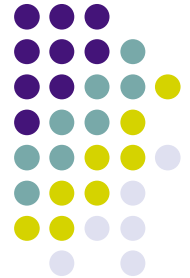
{ 1: MB1 ∨ MB2,
2: CPU1 ∨ CPU2,
3: SCSI ∨ IDE,
4: MB1 ⇒ ¬CPU2,
5: MB2 ⇒ ¬CPU1,
6: MB2 ⇒ ¬SCSI }

Query SRS: { SCSI, CPU2 },
i.e. is DD∪SRS consistent,
i.e. does DD |= ¬SRS not hold?

Compact notation of compiled theory:

{ 1+17: MB1 ⇔ ¬MB2,
2+14: CPU1 ⇔ ¬CPU2,
3: SCSI ∨ IDE,
4+16: MB1 ⇔ ¬CPU2,
5+13: MB2 ⇔ ¬CPU1,
6: MB2 ⇒ ¬SCSI,
7+11: CPU2 ⇔ MB2,
8+10: CPU1 ⇔ MB1,
9: SCSI ⇒ MB1, 12: MB2 ⇒ IDE,
15: CPU2 ⇒ ¬SCSI, 18: SCSI ⇒ CPU1,
19: ¬IDE ⇒ MB1, 20: CPU2 ⇒ IDE,
21: ¬IDE ⇒ CPU1
}

Knowledge Compilation: Example (cont´d)



Compiled theory DD_{comp}:

{ 1+17: MB1 ⇔ ¬MB2,
2+14: CPU1 ⇔ ¬CPU2,
3: SCSI ∨ IDE,
4+16: MB1 ⇔ ¬CPU2,
5+13: MB2 ⇔ ¬CPU1,
6: MB2 ⇒ ¬SCSI,
7+11: CPU2 ⇔ MB2,
8+10: CPU1 ⇔ MB1,
9: SCSI ⇒ MB1, 12: MB2 ⇒ IDE,
15: CPU2⇒¬SCSI, 18: SCSI⇒CPU1,
19: ¬IDE ⇒ MB1, 20: CPU2 ⇒ IDE,
21: ¬IDE ⇒ CPU1
}

In the compiled theory:

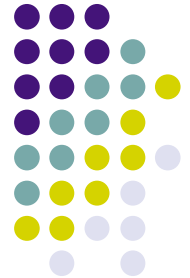
- Queries can be answered directly, e.g. { SCSI, CPU2 } violates constraint 15
- Queries under which all constraints evaluate to true can always be extended to valid configurations, e.g.

  { SCSI, CPU1, MB1 } or

  { IDE, CPU2, MB2 } or

  { SCSI }

Knowledge Compilation: Example (cont´d)



Compiled theory DD_{comp}:

{ 1+17: MB1 ⇔ ¬MB2,
2+14: CPU1 ⇔ ¬CPU2,
3: SCSI ∨ IDE,
4+16: MB1 ⇔ ¬CPU2,
5+13: MB2 ⇔ ¬CPU1,
6: MB2 ⇒ ¬SCSI,
7+11: CPU2 ⇔ MB2,
8+10: CPU1 ⇔ MB1,
9: SCSI ⇒ MB1, 12: MB2 ⇒ IDE,
15: CPU2⇒¬SCSI, 18: SCSI⇒CPU1,
19: ¬IDE ⇒ MB1, 20: CPU2 ⇒ IDE,
21: ¬IDE ⇒ CPU1
}

In the compiled theory:

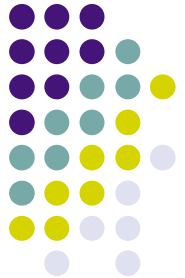
- Valid orders can be generated by unit propagation (domain reduction in the CSP sense):

  e.g. SRS = { MB1 } fixes the following variables:

  MB1=CPU1=true,
  MB2=CPU2=false

  Alternative view: Compiled theory simplifies to { MB1, ¬MB2, SCSI∨IDE, ¬CPU2, CPU1 }

Product Configuration



- Given a *domain description* DD and a *requirements specification* SRS [Felfernig:2000] we can:
  - Check a complete or partial order c represented by a set of literals for consistency:

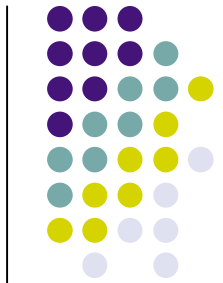
    c consistent $\Leftrightarrow$ $DD_{comp} \cup SRS \cup$ c consistent

    [1 entailment check in compiled theory]
  - Generate valid orders for consistent $DD_{comp} \cup SRS$:

    c valid $\Leftrightarrow$ c $\models$ $DD_{comp} \cup SRS$

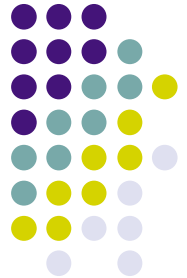
    [series of entailment checks,
    unit propagation to simplify DD_{comp}]
- No Backtracking required!

Knowledge Compilation Algorithms



- Generation of compiled knowledge base by *prime implicate (PI) computation*
  - Algorithms for PI computation: Tison´s algorithm [Tison:67] or BDD-based methods [Simon and del Val:2001]
  - Number of prime implicates may be exponential in the size of the knowledge base
- Other approaches:
  - Theory approximation
  - Combination of knowledge compilation with tractable inference algorithm
    - E.g. Unit-Resolution (UR) complete compilation

Theory Approximation

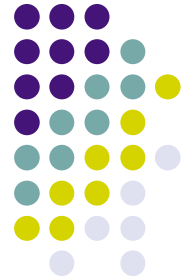


- Compute two computationally tractable theories approximating T from above (T_{ub}) and below (T_{lb})
- Then use the following algorithm:

  ALGORITHM THEORY-APPROX
  INPUT: T_{lb}, T, T_{ub}, c with $T_{lb} \models T \models T_{ub}$
  BEGIN
  IF $T_{ub} \models c$ THEN return true
  ELSE IF not ($T_{lb} \models c$) THEN return false
  ELSE output „don‘t know“
  END

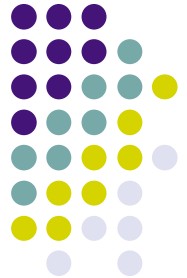
- Note: Computation of theories T_{ub} and T_{lb} may be hard.

UR Complete Compilation



- Combination of prime implicate computation (for knowledge compilation) with unit resolution (tractable inference algorithm)
  - Compute only those prime implicates that cannot be derived by unit resolution
  - Different algorithms for UR complete compilation by del Val [delVal:1994]
  - Novel algorithm computing different knowledge base DD_{comp*} presented in workshop proceedings
- Generates (at best exponentially) smaller knowledge bases than purely PI-based algorithms

Experimental Results

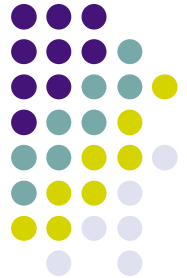


- Compilation of automotive product configuration data base for Mercedes-Benz cars:

model line	#prop. vars	|DD|	$\|DD_{comp}\|$	$\|DD_{comp^*}\|$
C250 FV	1465	2356	2492	1837
C210 FVF	1934	3985	496050800	--

- Time to generate compiled theory DD_{comp}: 93.5 sec (C250 FV) resp. 426.55 sec (C210 FVF) on a Pentium III

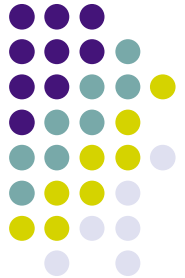
Conclusion



- ☺ Knowledge compilation suitable for processing huge series of configuration problems
- ☺ Compilation can deliver run-time guarantee for processing each configuration instance

- ☹ Computation of compiled knowledge base can be infeasible

Future Work



- Other knowledge compilation algorithms
- Further experiments
- Make use of variable orderings (⇢ ordered resolution)