Chapter 8

APPLICATIONS

8.1 KL-ONE type languages

8.1.1 KL-ONE and predicate logic

The knowledge representation system KL-ONE was introduced in [BS 85] and a large number of systems based on the similar ideas have been built, for example BACK [NvL 88], CLASSIC [BBMR 89], KANDOR [Pat 84], KL-TWO [Vil 85], KRYPTON [BPGL 85], LOOM [MB 87], NIKL [KBR86]. Generally speaking, KL-ONE type languages can be viewed as certain frame-based, database-oriented languages, where heavy stress is put on the definition of concepts, which can be viewed as unary predicates. Another common feature of these systems is the separation of the knowledge into a terminological part (definitions of concepts), called "T-box" and an assertional part (database), called "A-box". There is a special concept language for defining concepts in the T-box, which can be translated easily to (first order) predicate logic. Actually the concept language corresponds to a certain fragment of predicate logic.

EXAMPLE 8.1.: Given the concepts "person", "female" and "shy", the concept "persons who are female or not shy (pwafons)" can be expressed by the formula

pwafons = (and person (or female (not shy)))

in concept logic, with the translation

$(\forall x)\,\mathrm{pwafons}(x) \Leftrightarrow (\mathrm{person}(x) \wedge (\mathrm{female}(x) \vee \neg\,\mathrm{shy}(x)))$

in predicate logic.

The user of the language can describe concepts in the T-box, write down known facts in the A-box, and ask queries from the system, similarily like it is done in PROLOG. Differently from PROLOG, however, the language is decidable - any query can be answered (either positively or a negatively) in finite time. Several other important properties are also required to be decidable. For example, the question whether a T-box is satisfiable, or the question whether one concept P is a subconcept of another concept Q. The last question

is called "subsumption problem" in the KL-ONE context: the problem can be read as "are all objects of type P also of type Q?". For example, given a definition of "female" as a "person" who is not "man", the concept "female" is subsumed by the concept "person". Several KL-ONE type systems have the capability to compute the full tree defined by the relation of concept subsumption, and this capability is considered to be rather important from the practical point of view.

So far no language exists which could be called "the KL-ONE language" - every system implements a language with different restrictions and different extensions. However, the language called ALC which has been introduced in [SS88] can be considered as a basic language for KL-ONE systems, on which different extensions can be built. In [SS88] it is shown that satisfiability and KL-ONE-subsumption in ALC are PSPACE-complete.

ALC contains two kinds of basic symbols, denoting concepts (corresponding to unary predicates) and roles (binary predicates). ALC is intended as a media for writing sets of concept definitions. We review a definition of the formulas of ALC, called concept description (compare e.g. [BBHNS90]). Simultanously, we define for every concept description of ALC a corresponding formula F' of predicate logic, called translation of F:

DEFINITION 8.1: Any concept description of ALC has one of the following forms:

- ***TRUE*** - top concept of the subsumption hierarchy. Translation: ***TRUE*** ≡ the truth value "true" (or some tautology).

- ***FALSE*** - bottom concept of the subsumption hierarchy. Translation: ***FALSE*** ≡ the truth value "false" (or some contradiction).

- ***P***, where ***P*** is a concept symbol (representing an atomic concept). Translation: P' ≡ P(x), where P is a predicate symbol uniquely corresponding to P; x, is a free variable.

- ***(not F)***, where ***F*** is an arbitrary concept description of ALC. Translation: $\neg F'$, where ***F'*** is the translation of ***F***.

- ***(and F G)***, where ***F*** and ***G*** are arbitrary concept descriptions. Translation: $(F' \wedge G')$, where ***F'*** and ***G'*** are the translations of ***F*** and ***G***, respecitively.

- ***(or F G)***, where ***F*** and ***G*** are arbitrary concept descriptions. Translation: $(F' \vee G')$, analogously to conjunction.

- ***(all R F)***, where ***R*** is a role, and ***F*** is an arbitrary concept description. Translation: $(\forall y)(R(x,y) \Rightarrow F'[y])$, where x is a (new) free variable and F' contains no free variables except y. (This is what the notation ***F'[y]*** indicates).

- ***(exists R F)***, where R is a role, and ***F*** is an arbitrary concept description. Translation: $(\exists y)(R(x,y) \wedge F'[y])$, where x is a (new) free variable and ***F'*** contains no free variables except y.

DEFINITION 8.2.: A concept definition in ALC is an expression of the following form: ***P = F***, where ***F*** is a concept description and ***P*** a concept symbol. Translation: $(\forall x)(P(x) \Leftrightarrow F'[x])$.

A very important additional restriction to the described language is that no circularities are allowed in concept definitions - that is, there must be a certain hierarchy of concepts such that all concepts in the right side of any concept definition are lower w.r.t. the hierarchy than the defined concept. For example, it is not allowed to define a human as an animal whose mother is human: ***human = (and animal (exists mother human))***, since this definition contains a circularity.
Now, considering the properties of satisfiability and universal satisfiability of concept descriptions as defined e.g. in [BBHNS90], we may state the following:

Proposition 8.1.: A concept description D is satisfiable iff the first order formula $((\forall x)P(x) \Leftrightarrow D'[x])$ and $(\exists x)P(x)$ is satisfiable, where D'[x] is the translation of

D and P is predicate symbol not occurring in D'[x]. Moreover, D is universally satisfiable iff $((\forall x)P(x) \Leftrightarrow D'[x])$ And $(\forall x)P(x)$ is satisfiable.

We will present a sample T-box given by F. Baader.

female = (not male)
person = (some sex (or male female))
parent = (and person (some child person))
mother = (and parent (some sex female))
father = (and parent (not mother))
randparent = (and parent (some child parent))
parent–with–sons–only = (and parent (all child (some sex male)))

The following is a translation of this T-box into predicate logic:

$(\forall x)$ (female(x) $\Leftrightarrow$ $\neg$ male(x))

$(\forall x)$ (person(x) $\Leftrightarrow$ $(\exists y)$(sex(x,y) $\wedge$ (male(y) $\vee$ female(y))))

$(\forall x)$ (parent(x) $\Leftrightarrow$ person(x) $\wedge$ $(\exists y)$(child(x, y) $\wedge$ person(y)))

$(\forall x)$ (mother(x) $\Leftrightarrow$ parent(x) $\wedge$ $(\exists y)$(sex(x, y) $\wedge$ female(y)))

$(\forall x)$ (father(x) $\Leftrightarrow$ parent(x) $\wedge$ $\neg$ mother(x))

$(\forall x)$ (grandparent(x) $\Leftrightarrow$ parent(x) $\wedge$ $(\exists y)$(child(x, y) $\wedge$ parent(y)))

$(\forall x)$ (parent–with–sons–only(x) $\Leftrightarrow$ parent(x) $\wedge$

$\wedge$ $(\forall y)$(child(x, y) $\Rightarrow$ $(\exists z)$(sex(y, z) $\wedge$ male(z))))

8.1.2 Clause forms of ALC definitions

As the ALC-language allows arbitrary nesting of quantifiers, it is not immediately clear whether it corresponds to a (subset of) some well-known decision class. However, due to the fact that any subformula starting with a quantifier contains at most one free variable, the following operator *dissect* converts any predicate logic form of an ALC- definition to a new formula, belonging to a decidable class closely related with Gödel's Class (i. e. the class with prefix $\exists^*\forall\forall\exists^*$):

DEFINITION 8. 3.: Let F be the translation of some ALC-definition. Let G[x] be some subformula of F s.t. G starts with a quantifier which is in the scope of two other quantifiers. (Remember that the notation G[x] indicates that G contains a single free variable x). Then *dissect*(F) ≡ $(F^{G}_{P(x)} \wedge (\forall x)(P(x) \Leftrightarrow G[x])$, where $F^{G}_{P(x)}$ is the result of replacing in F the subformula G with the atom P(x) where P is a new predicate symbol. If there is no subformula of the indicated kind in F then let dissect(F) ≡ F.

We call a formula F^{*} that results from iterated applications of dissect to a formula F simplified form of F iff dissect(F^{*}) = F^{*}.

Obviously, there is no unique simplified form in general. But the property we are interested in is captured by the following:

Proposition 8. 2.: A simplified form F^{*} of a translation F of some ALC-definition is satisfiable iff F is satisfiable.

In the sections to come we will use clause sets corresponding to simplified forms of translations of ALC-definitions to demonstrate that questions concerning (universal) satisfiability and subsumption of concepts are decidable using resolution methods.

As an example, we will present a translation of the last formula from our first sample T-box (the other formulas remain unchanged):

$(\forall x)(\text{parent\_with\_sons\_only}(x) \Leftrightarrow \text{parent}(x) \wedge (\forall y)(\text{child}(x,y) \Rightarrow P_1(y)))$
$(\forall y)(P_1(y) \Leftrightarrow (\exists z)(\text{sex}(y,z) \wedge \text{male}(z)))$

The following set *KL-ONE-example* is a clause form of the translated T-box above:

comment: definition of female:

$\{\neg \text{female}(x),\ \neg \text{male}(x)\}$
$\{\text{male}(x),\ \text{female}(x)\}$

comment: definition of person:

{¬person(x), sex(x,f(x))}

{¬person(x), male(f(x)), female(f(x))}

{¬sex(x,y), ¬male(y), person(x)}

{¬sex(x,y), ¬female(y), person(x)}

comment: definition of parent:

{¬parent(x), person(x)}

{¬parent(x), child(x,g(x))}

{¬parent(x), person(g(x))}

{¬person(x), ¬child(x,y), ¬person(y), parent(x)}

comment: definition of mother:

{¬mother(x), parent(x)}

{¬mother(x), sex(x,h(x))}

{¬mother(x), female(h(x))}

{¬parent(x), ¬sex(x,y), ¬female(y), mother(x)}

comment: definition of father:

{¬father(x), parent(x)}

{¬father(x), ¬mother(x)}

{¬parent(x), mother(x), father(x)}

comment: definition of grandparent:

{¬grandparent(x), parent(x)}

{¬grandparent(x), child(x,k(x))}

{¬granparent(x), parent(k(x))}

{¬parent(x), ¬child(x,y), ¬parent(y), grandparent(x)}

comment: definition of parent–with–sons–only (abbreviated as pwso):

{¬pwso(x), parent(x)}

{¬pwso(x), ¬child(x,y), p1(y)}

{¬parent(x), child(x,l(x)), pwso(x)}

{¬parent(x), ¬p1(l(x)), pwso(x)}

{¬p1(y), sex(y,r(y))}

{¬p1(y), male(r(y))}

{¬sex(y,z), ¬male(z), p1(y)}

Notice that the descriptive power of a T-box above is more limited than one might expect. For example, given the following translation of an A-box

{person(John), child(John,Peter), sex(Peter,Male), male(Male)},

it is not possible to deduce pwso(John). In order to deduce pwso(John), we would need a fact $(\forall x)(\text{child}(\text{John},x) \Rightarrow x = \text{Peter})$. Unfortunately we do not have the equality predicate in the language!

Some experiments with the clause set above using an implementation of the k-refinement of chapter 4 are described in section 9.2.2 below.

8.1.3 Extensions of ALC

Consider the following natural extension (we call it *One-free*) of the class of formulas that are translations of ALC-definitions or descriptions.

DEFINITION 8.4 .: Any formula F of the predicate logic without the equality predicate and without function symbols belongs to the class *One-free* iff any subformula of F starting with a quantifier contains at most one free variable.

Recall that we do not consider the constant symbols to be functional - with other words, we allow formulas in *One-free* to contain constant symbols.
The operator *dissect* obviously remains sound (w.r.t. to satisfiability) for class *One-free* and guarantees that no subformula of a simplified formula is in the scope of more than two quantifiers. Thus the resulting formulas are closely related with the formulas in the Gödel class (cf. chapter 5 or 6). We are not exactly concerned with the Gödel class, since the prenex form of a simplified formula may have an arbitrarily deep quantifier nesting.

However, it is easy to see that we may transform any simplified formula of class One-free to a set S of clauses s.t. for all $C \in S$: Either

(i) $|V(C)| \leq 2$ and C is function free, or
(ii) C contains only one variable x and at most unary function symbols all of which contain x as argument.

Observe that the resulting class of clause sets is a simple subset of class $\mathcal{S}^+$, described in chapter 5. Therefore we may use the resolution refinement R_m (see section 5.4.1 and 5.4.2) as a decision procedure for this class. Some simple additional observations thus imply that resolution methods may be employed to decide various important properties of ALC-definitions, like (universal) satisfiability and subsumption of concepts. If S is the clause form corresponding to the concept defintions of some A-box, $R_m^*(S)$ may be used as a basis to anwser efficiently whole series of questions about this A-box.

To mention another resolution variant that is investigated in this monography we remark that also the termination of the k-refinement of chapter 4 (or $>_{sv}$-refinement of chapter 5) on this class is easy to show: One has to modify the termination proof for Gödel's Class. An important point to notice is that any clause derived from a clause obeying (i) above and a clause obeying (ii) is splittable into clauses obeying (ii). Analogously to the Gödel class, there is no need for actual splitting during proof search - using Robinson's resolution or corresponding restriction on factorization is enough.

As for completeness, the same remarks hold as for Gödel's class: to preserve lifting, subsumption must be restricted in the following way: a literal P(x, y) must not be allowed to subsume a literal P(x, x){t/x} for any t. Tautology-elimination must be restricted correspondingly:

{P(x,x), ¬P(x,x)} {t/x} should not be eliminated for any t, while eliminating {P(x,y), ¬P(x,y)} is OK.

8.1.4 Functional relations

Some extended versions of the ALC language (the one presented in [HN 90], for example) also allow to declare certain relations to be functional (e.g., the relation "sex" in the example above would naturally be defined as a functional relation). The predicate logic equivalent of declaring a relation (say, R) to be functional on the first argument would be a following axiom:

$$(\forall x y z)\ ((R(x,y) \wedge R(x,z)) \Longrightarrow (y = z))$$

The axiom uses the equality predicate and thus we would have to handle the defining properties of equality e.g. by explicit axiomatization of equality, that is introduction of axioms for reflexivity, commutativity, transitivity and axioms for substitution into atoms. Including the full axiomatization for equality would mean that the resulting formulas are not any more members of known decidable classes of predicate logic, thus making the translation useless for our purposes.

In principle functionality of a binary relation can be naturally expressed using one-place function symbols instead of two-place predicate symbols combined with the functionality axiom. This means that instead of using a quantifier to introduce a new functionally bound variable, we can use a term f(x), where f is a function symbol representing the functional relation and x depends on the context.

So, we could consider extending class *One-free* by allowing one-place function symbols. Unfortunately, this extension turns out to be undecidable, as it contains the $\forall\exists\forall$-Class.
We could consider ways of imposing additional restrictions to the extended *One-free* class. One rather strong restriction, which we will shortly investigate in the following, is obtained by disallowing any non-functional relations. That is, we require that all binary relations are functional - then they can be represented by one-place function symbols and we can restrict the class to contain only monadic predicate symbols. The resulting class, of course, is not any longer an extension but rather a variant of class *One-free*.

It is natural to extend this new class in a way s.t. it corresponds to class E (see chapters 4 and 5).

DEFINITION 8.5.: Any formula F of the predicate logic without the equality predicate, but possibly containing function symbols belongs to the Class *One-free-E* (OFE, for short) iff any subformula of F starting with a quantifier contains at most one free variable and all atoms contain at most one variable.

Obviously the simplified form (using the operator simple) of any formula in class OFE belongs to the Class E, the decidability of which has been shown in various ways in chapters 4 and 5.

Class OFE can be used as a media to express questions about both, the T-box and the A-box. As for the A-box, we also allow ground equality units such that the set of all equality units can be oriented to a complete (confluent and terminating) set of term rewriting rules (cf. [KBR86]. Thus all the facts of the form R(a,b) for any functional R can be translated to equality units f(a) = b.

In order to decide any formula F in the OFE Class containing also ground equality units, do the following:

- Convert F to a simplified form F^* (using the operater *dissect* presented in the last section).
- Convert F^* to a clause set S (equivalent to F^* w.r.t. satisfiability).
- Use the Knuth-Bendix algorithm to check the completeness of the set of equality units; if needed, try to complete the set. Notice that on the ground set of equality units the Knuth-Bendix completion algorithm will always terminate.
- Use narrowing to remove the rewrite rules resulting from the previous step - this gives a fully narrowed clause set S' which is in class E. (cf. [Sla74] and chapter 8 to see the definition and examples of narrowings.)
- Use techniques from chapter 4 or 5 to decide the clause set S'.

As an example we take the previous T-box, assume the relation "sex" to be declared functional and translate the selected part to OFE. Unfortunately, the OFE Class is too restrictive to translate the whole of the previous T-box in a "coherent" way. The reason here is that the relation child cannot be considered to be functional on the first argument. We will use the notion of "first child" instead: ***first–child*** is assumed to be functional. Whenever some person x does not have children, we can define first–child(x) = $\perp$, for some constant $\perp$ for which we know that $\neg$person($\perp$). In the following T-box we have replaced some equivalences by implications - the sole reason being that we felt the implications to correspond better to our intuitive understanding of the presented terminology.

$\neg$ person($\bot$)

female(Female)

male(Male)

$(\forall x)$ (female(x) $\Rightarrow$ $\neg$ male(x))

$(\forall x)$ (person(x) $\Rightarrow$ male(sex(x)) $\vee$ female(sex(x)))

$(\forall x)$ (parent(x) $\Leftrightarrow$ person(x) $\wedge$ person(first–child(x)))

$(\forall x)$ (mother(x) $\Leftrightarrow$ parent(x) $\wedge$ female(sex(x)))

$(\forall x)$ (father(x) $\Leftrightarrow$ parent(x) $\wedge$ $\neg$ mother(x))

As for the concepts *grandparent* and *parent–with–sons–only*, we could assume that the number of children a person can have is bounded by some N and use the following clumsy translation:

$(\forall x)$ (grandparent(x) $\Leftrightarrow$ parent(x) $\wedge$

(parent(first–child(x)) $\vee$ ... $\vee$ parent(N^{th}–child(x))))

$(\forall x)$ (man(x) $\Leftrightarrow$ (person(x) $\wedge$ male(sex(x))))

$(\forall x)$ (parent–with–sons–only(x) $\Leftrightarrow$ parent(x) $\wedge$

$\bigwedge_{i=1}^{N}$ person(i^{th}child) $\Rightarrow$ male(sex(i^{th} child)))

Now we could, for example, add a sample A-box to the given T-box (John, Peter and Male are constant symbols):

first–child(John) = Peter

second–child(John) = $\bot$, ... , N^{th}–child(John) = $\bot$

sex(first–child(John)) = Male

grandparent(John)

Completion will add a new equality unit sex(Peter) = Male to the ones above. Now it is easy to check out that from the full narrowing of the clause form of the last T-box above it is possible to derive e.g. father(Peter) without using the equality units any more. The following clauses from the fully narrowed set are used for this derivation:

{$\neg$ person($\bot$)}

{male(Male)}

{$\neg$ female(x), $\neg$ male(x)}

{$\neg$ mother(Peter), female(Male)}

{$\neg$ parent(x), mother(x), father(x)}

{¬grandparent(John), parent(Peter), ... , parent(⊥)}
{¬parent(x), person(x) }
{grandparent(John)}

8.2 EXPERIMENTS WITH THE IMPLEMENTATION

In order to experiment with the decision refinements of resolution, we have written a resolution theorem prover containing various well-known and less well-known search strategies and refinements. The prover (see [Tam 90], [Tam 91]) is implemented on IBM-PC in muLISP; unification and subsumption are coded directly in assembly. The prover is suitable both for fully automatic and interactive use.

The prover incorporates a special part for building finite models. This part does not implement the complete method described in chapter 8, but a "minimal consistent criteria" method with no backtracing. We do not have a proof that such a simple method always terminates on Class AM; neither have we any counterexamples. The minimal-criteria method has performed quite well in the experiments. Although we know that it does not terminate on all formulas from Gödel's Class, it has been able to build finite models for several satisfiable formulas in Gödel's Class presented in [Chu56].

The prover has been used as an aid for solving an open problem: condensed detachment completeness of relevance logic (see [MT 91]). However, decision strategies were not used for this application and thus we won't give any details here.

All the following experiments have been performed on the 640-Kbyte machine with the 80286 processor running at 16 MHZ - a rather small and slow machine by current standards.

8.2.1 Hyperresolution-based methods

In chapter 3 semantic clash refinements where used to decide the classes PVD and OCC1N. The essential feature of the decision method was the algorithmical selection of an appropriate setting according to the syntactical

structure of the set of clauses. We will show that the decision method for PVD is also an efficient theorem prover.

Consider first the set of clauses

$$
\begin{aligned}
H_1 = \{ & \{ \neg P(x,y,z),\ P(z,y,x)\} \\
& \{\neg P(x,y,z),\ P(y,x,z)\} \\
& \{\neg P(x,y,z),\ P(x,y,g(z))\} \\
& \{\neg P(x,y,z),\ P(x,y,f(z))\} \\
& \{P(a,b,c)\} \\
& \{\neg P(f(g(a)),\ f(g(b)),\ f(g(c)))\}\ \}
\end{aligned}
$$

It is easy to see that H_1 is in PVD via the positive setting M_p. Using negative hyperresolution (i.e. computing $R^*_{M_p}(H_1)$) the empty clause was derived in 14 seconds on level 12, 162 derived clauses were retained. By "retained" we mean "not immediately eliminated as tautologies or subsumed clauses". Thus, clauses eliminated by back subsumption are considered to be "retained".

Positive hyperresolution ran out of space at level 7, after retaining 1433 clauses in 450 seconds. The reason for the extremely different behaviour of positive and negative hyperresolution is the following: While negative hyperresolution reduces the depth of terms, positive hyperresolution does the contrary. The resolution based decision procedure for PVD, defined in chapter 3, automatically selects the positive setting and gets a fast refutation of H_1.

The following set H_2 is slightly more complex than H_1, but expresses a similiar mathematical problem.

$$
\begin{aligned}
H_2 = \{ & \{P(x,y,z,u),\ \neg P(u,x,y,z)\} \\
& \{P(x,y,z,u),\ \neg P(u,z,y,x)\} \\
& \{P(x,y,z,u),\ \neg P(f(x),y,z,u)\} \\
& \{P(x,y,z,u),\ \neg P(x,y,z,g(u))\} \\
& \{\neg P(a,b,c,d)\} \\
& \{P(f(g(d)),f(g(c)),f(g(b)),f(g(a)))\}\}
\end{aligned}
$$

H_2 also belongs to Class PVD, but via negative setting. We observe an effect similiar to that for H_1: By positive hyperresolution the refutation was found in 120 seconds on the 13^{th} level. 645 clauses were retained. Negative hyperresolution ran out of space at level 6, after retaining 1339 clauses in 411 seconds.

While H_1 is in KII, H_2 is in KI (see chapter 3 and [Lei 90]); but both, H_1 and H_2, are in PVD (by KI $\cup$ KII $\subset$ PVD). While positive hyperresolution decides KI, negative decides KII, but no fixed setting refinement decides both of them. Moreover, by the experiments discussed before, we see that neither positive nor negative hyperresolution is suited for refuting both, H_1 and H_2. Thus the clause sets H_1,

H_2 illustrate the value of using resolution decision procedures as "ordinary" theorem provers: besides extension of the knowledge about the decision problem, finding larger decidable classes leads to better theorem proving programs. That a "decider" yields a good theorem prover is by no means surprising, because many resolution based decision procedures (as the one described for PVD) do not increase the term depth of clauses and thus keep complexity low.

In general it will pay out to build an expert system, which - given a set of clauses S - tries to find a (predefined) decision class Γ s.t. $S \in \Gamma$; afterwards the decision method is used to refute S. By the undecidability of predicate logic such a method cannot always succeed, but may be of practical value nevertheless.

Finally we define a set of clauses H_3 which is not in PVD (although again "similiar" to H_1 and H_2).

$H_3 = \{\{\neg P(x, y, z,), \neg P(y, u, z), P(x, u, z)\}$
$\{\neg P(x, y, z), P(z, y, x)\}$
$\{\neg P(x, y, z), P(y, x, z)\}$
$\{\neg P(x, y, z), P(x, y, f(z))\}$
$\{\neg P(x, y, z), P(g(x), y, z)\}$
$\{P(a, f(b), c)\}$
$\{P(f(b), d, c)\}$
$\{\neg P(g(f(a)), g(f(d)), g(f(c)))\}\}$

Without the first clause C = {¬P(x, y, z), ¬P(y, u, z), P(x, u, z)} H_3 would be in PVD via positive setting. But for M_p we get $V(C_{neg}) - V(C_{pos}) \neq \emptyset$, which violates PVD2. On the other hand, taking the negative setting M_n and the clause D = {¬ P(x,y,z), P(x,y,f(z))} we get $\tau_{MAX}(x, D_{neg}) > \tau_{MAX}(x, D_{pos})$, which again violates PVD2. Thus no set of clauses containing C as well as D can be in PVD. It is significant that both, positive hyperresolution and negative hyperresolution ran out of space in the attempt to refute H_3. Moreover, it is easy to verify that also A-ordering refinements fail to refute H_3 within reasonable time and space bounds. In fact H_3 is not contained in any decidable class investigated in this monography. However, it is not hard to find a hyperresolution refutation of depth 14 by hand (transitivity and permutations are easily handled by human problem solvers, but not by standard resolution theorem provers).

8.2.2 Ordering refinements

[Chu 56] presents a large set of formulas in several decidable classes, meant as exercises for the reader. Some of these formulas are satisfiable, some are not. Some of the formulas are very easy to decide, some are quite hard.

The following table presents the results of using the k-refinement for deciding all the formulas from [Chu 56]. Each row of the table contains the number of the formula (in the set of exercises to section 46 of [Chu 56]), its prefix, time of deciding a formula, number of derived clauses, number of retained clauses, result found (negation of the formula satisfiable-/unsatisfiable), time of constructing a finite domain, size of the domain found. All times are given in seconds (shorter times are somewhat inaccurate). '***' means that the finite model was not found (due to memory limitations or inherent incompleteness of the used model-building method).

Ex2 No1	EAE	0.11	3	2	unsat.	-	-
Ex2 No2	AE	0.06	1	0	sat.	0.05	2
Ex2 No3	EEEA	0.02	0	0	sat.	0.06	1
Ex2 No4	EEA	0.11	1	1	unsat.	-	-
Ex2 No5	AAEE	0.38	13	9	unsat.	-	-
Ex3 No1	EAEE	0.05	0	0	sat.	0.17	2

Ex3 No2	EEEAA	0.11	3	2	unsat.	-	-
Ex4 No1	EA	0.05	2	1	sat.	0.04	1
Ex4 No2	EA	0.17	3	2	unsat.	-	-
Ex9 No1	AE	0.17	8	2	sat.	0.55	2
Ex9 No2	AE	0.22	5	5	unsat.	-	-
Ex12 No1	AEE	0.33	6	6	unsat.	-	-
Ex12 No2	EAE	0.33	16	5	unsat.	-	-
Ex12 No3	AEE	1.93	112	40	unsat.	-	-
Ex14 No1	AAE	0.06	0	0	sat.	4.5	7
Ex14 No2	AAE	2.69	196	20	sat.	***	***
Ex14 No3	AAE	0.22	3	3	unsat.	-	-
Ex14 No4	AAE	2.75	214	30	unsat.	-	-
Ex14 No5	AAE	0.28	5	5	unsat.	-	-
Ex14 No6	AAE	1.43	93	21	unsat.	-	-
Ex14 No7	AAE	0.55	19	5	sat.	***	***
Ex15 No1	EAAE	0.11	0	0	unsat.	-	-
Ex15 No2	EEAE	0.11	0	0	sat.	0.05	2
Ex15 No3	AEA	0.06	0	0	unsat.	-	-
Ex15 No4	AEA	0.06	0	0	unsat.	-	-
Ex15 No5	AEAA	0.05	0	0	unsat.	-	-
Ex15 No6	AEAA	0.05	0	0	unsat.	-	-
Ex15 No7	EAEA	0.06	0	0	sat.	0.06	2
Ex16 No2	EAAE	0.06	0	0	unsat.	-	-
Ex16 No3	EAAE	0.11	1	1	sat.	0.17	2
Ex16 No4	EEAAE	0.72	9	5	unsat.	-	-
Ex17 No2	EEAAEE	0.11	1	1	unsat.	-	-
Ex17 No3	EEAAE	1.32	42	34	unsat.	-	-
Ex17 No4	AEAE	30.5	3182	0	sat.	***	***
Ex17 No5	AEAE	7.58	588	22	unsat.	-	-
Ex18 No2	EAAE	1.43	98	10	unsat.	-	-
Ex18 No3	EEAAE	2.04	140	33	sat.	8.79	4
Ex18 No4	EEAAE	0.66	29	8	unsat.	-	-
Ex18 No5	EAAAE	15.16	654	109	unsat.	-	-
Ex20 No1	EEAAE	0.66	28	11	unsat.	-	-

An important point noticed during experimentation is that the decision refinement performed very well (when compared to any other known resolution strategy) for the unsatisfiable formulas in [Chu 56]. Consider the hardest unsatisfiable formula in [Chu 56]: Ex18 No5. The following is the clause form of Ex18 No5:

{¬P(x,y,z), ¬P(a,a,f(x,y,z)), P(y,z,x), P(z,x,y)}
{¬P(z,x,y), P(x,y,z), ¬P(y,x,f(x,y,z))}
{¬P(z,x,y), P(y,z,x), ¬P(y,x,f(x,y,z))}
{P(z,x,y), P(y,x,f(x,y,z))}
{¬P(x,y,z), ¬P(y,z,x), P(y,x,f(x,y,z))}
{¬P(y,z,x), P(x,y,z), ¬P(x,f(x,y,z),y)}
{¬P(y,z,x), P(z,x,y), ¬P(x,f(x,y,z),y)}
{P(y,z,x), P(x,f(x,y,z),y)}
{¬P(x,y,z), ¬P(z,x,y), P(x,f(x,y,z),y)}
{P(z,x,y), P(x,y,z), ¬P(f(x,y,z),y,x)}
{P(y,z,x), P(x,y,z), ¬P(f(x,y,z),y,x)}
{¬P(z,x,y), ¬P(y,z,x), P(f(x,y,z),y,x)}
{¬P(x,y,z), P(f(x,y,z),y,x)}
{P(x,y,z), P(f(x,y,z),f(x,y,z),f(x,y,z))}
{P(y,z,x), P(f(x,y,z),f(x,y,z),f(x,y,z))}
{P(z,x,y), P(f(x,y,z),f(x,y,z),f(x,y,z))}
{¬P(x,y,z), ¬P(y,z,x), ¬P(z,x,y), ¬P(f(x,y,z),f(x,y,z),f(x,y,z))}

The derivation of the empty clause found by the prover was 26 levels deep. With no other well-known strategy of resolution was the prover able to show unsatifiability of this clause set. We experimented also with W. McCune's prover OTTER, and this prover was also unable to find the refutation. Of course, all the experiments were performed on a rather small machine. It would be interesting to see how powerful a machine is needed to find the refutation of the clause set above using some proof strategy sufficiently different from the ordering refinement we used.

8.2.2 Experiments with the translations of KL-ONE formulas

Recall the example clause set *KL–ONE–example* from the above section describing the KL-ONE type languages. Our implementation of the k-refinement was able to show the satisfiability of this set in 10 seconds. The whole derivation was 8 levels deep, 320 clauses were derived and 70 of those were retained. The program was also successful to find out that this clause set has a finite model with an 1-element domain.

For this particular example the satisfiability follows from the propositional structure already and positive hyperresolution is unable to derive a single new clause from the whole set. However, the k-refinement did not use this fact. Also, it can be considered rather important to find the "completed set" and the whole KL-ONE-subsumption structure for the formula. The last two tasks were performed during those 10 seconds of work of the prover.