$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Paraconsistency and Programming

- This introduction has covered several fields in which paraconsistency is explored and developed. We have seen its merits, but also some problems and unsolved questions in these areas.
- Paraconsistent logics are more and more becoming an acceptable tool of work in the formal sciences and in modelling in the cognitive sciences and philosophy. With that development the fields of application are increasing.
- Here we look at an aera that is often mentioned as one of the main areas of future – commercial? – applications of paraconsistency: Data-management and programming.
- The focus here lays on weak paraconsistency, since the contradictions we find in a database are supposedly due to our imperfect ways of gathering data about a non-contradictory part of the world.
- [Since otherwise we would have to go into details of logic and database programming that are beyond the scope and intent of this introduction this chapter only presents the basic points and refers you to the literature.]

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Databases

- Databases are sets of data that change with time, typically because new data are entered or old data are updated. Further on procedures programmed to execute (regularly) on the database change data in other areas/tables accordance with the new data.
- Depending on the organization developers invested in a database the integrity of the data is defined first (according to the standards of normalization used in standard data servers like Microsoft SQL-Server or Oracle or Sybase) and hopefully kept with procedures and dependency constraints that fire in case new data may affect the database integrity. This is far from trivial and can amount to enormously large code for large databases. These integrity constraints and the respective procedures *define* what counts as inconsistent data.
- Notwithstanding the cautious design of the database and the amount of integrity checking usually sooner or later the problem of inconsistent data arises.
- Nevertheless a typical database (server) will not deduce anything from inconsistent data, for example fill tables now with arbitrary entries.

p ∧ ¬p

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Databases (II)

- The entries computed by stored procedures running on an inconsistent database typically make sense although integrity and consistency are corrupted in the database.
- Factually the logic of such databases is not explosive.

- In case you know SQL, consistency keeping comes down to the typical normalization constraints and checking inserts or updates, as in:

```
CREATE TABLE employee (
emp_id int NOT NULL PRIMARY KEY CHECK
     (emp_id BETWEEN 0 AND 1000),
emp_name varchar(30) NOT NULL CONSTRAINT no_nums CHECK
     (emp_name NOT LIKE '%[0-9]%'),
mgr_id int NOT NULL REFERENCES employee(emp_id),
entered_date datetime NULL CHECK
     (entered_date >= CURRENT_TIMESTAMP),
entered_by int CHECK (entered_by IS NOT NULL),
     CONSTRAINT valid_entered_by CHECK (entered_by =
     SUSER_ID(NULL) AND entered_by <> emp_id)
)
```

Revision of Databases

- Even if the inconsistency of a database is seen as something that should be overcome immediately be revision of the relevant parts of it one needs some form of paraconsistent reasoning during or modelling the revision process itself.
- All – non-trivial – consequences of the inconsistent data should be present to identify the desirable ones.
- In fact revision algorithms often possess a higher computational complexity than paraconsistent logic programming does (which can be of polynominal complexity [cf. (Damasio 1996, Chap. 10)].

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Resolution

- Queries against data sets (in a language like PROLOG) or queries in automated theorem proving are typically answered by *resolution*. Resolution works by *Horn clauses*. Horn clauses are right to left conditionals: their *head* states what can be derived given we have what is said in their *body*. A query aks whether some statement is derivable. Resolution now checks whether this statement is the head of some Horn clause and if it finds one tries to work out whether it can derive the body of that clause (and so on with this new *goal* until the derivation is founded in some fact statement).
- Now, *ex contradictione qoud libet* does not apply in resolution in as much as there is not in general a sequence of steps by which an arbitrary query can be satisfied from an inconsistent theory/database.

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Negation as Finite Failure

- Negation (in a language like PROLOG) is often taken as the absence of positive evidence. So if one queries for a statement and no derivation (either from basic data or by resolution) is available the answer is "No". If the language's dialect involves a negation sign and one queries for the negated statement the answer will be "Yes". Thus a negation is derivable as finite failure to derive the unnegated statement.
- Negation as finite failure is obviously non-monotonic: Once you add something the statement queried for before may now become derivable.
- Negation as finite failure does not lead to an application of *ex contradictione qoudlibet*.

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Negation as Finite Failure (II)

- Negation as finite failure, however, means that typically one has only positive left hand sides of Horn clauses. One can circumwent this by inventing negative predicates like "`non_coloured(X)`" but this is not an *in-built* negation (operator), but depending on our understanding of this predicate. For these predicates – depending on their definition and the database – one can, of course, have `coloured(apple)` and `non_coloured(apple)` thus simulating real (strong) negation.
- For a better capturing of real live examples involving strong negation the proof engine (or programming language) should be extended by an in-built strong negation, that can also occur on the left hand side of a Horn clause [cf. (Damásio 1996)]. This negation is monotonic.
- Whereas negation as finite failure is merely non-explosive (and thus paraconsistent) the addition of a strong negation allows for arriving at *proven* contradictions: `A,not-A`.

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Paraconsistency as Meta-Theory

- Since even before the advent of paraconsistent logics the factual behaviour of – at least a great many – databases and proving techniques was paraconsistent one simply needs some version of paraconsistent logic to adequately *describe* what logic is really implemented in them.
- Paraconsistent Logic has a job as meta-theory of (database) programming here.
- The question is whether the paraconsistent *behaviour* of programmed systems is due to them following some kind of non-explosive logic or is due to them just working on restricted – supposedly consistent – sub areas of the system/the data.
- Resolution can work very well in an inconsistent theory, since proving the goal works its way only through subject *related* parts of the theory, so that some other part of the theory may well be inconsistent without ever coming into contact with the resolution going on.

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Procedural Paraconsistency

- In the light of programming one may give a *procedural* definition of what paraconsistency comes down to here. Whereas with respect to logical systems one considers the set of consequence as given right – immediately one might say – with the premises with respect to programs drawing on data one may consider of the consequences as delivered piecemeal.
- One can define then [cf. (Decker 2005)], given some inconsistent database T a distinction between non-trivializing procedures and those that do not exploit any occurring inconsistencies for new derivations:

> A (resolution) proof procedure Π is *somewhat paraconsistent* if there is a well formed formula A such that there is no proof of A, given T, using Π.
>
> A (resolution) proof procedure Π is *fully paraconsistent* if for each well formed formula A which has no proof in a consistent part T* of T there is no new proof in T of A using Π.

Some resolution algorithms are fully paraconsistent in this sense.

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

GVII

- Ginsberg has developed a 7-valued paraconsistent logic, called **VII**, that combines paraconsistency with non-monotonicity.
- The truth values are: true, true by default, false, false by default, indetermined, indetermined by default, contradictory. The designated truth values are: true, and contradictory.
- There is a strong negation meaning that there is positive evidence for the falsity of a statement, and there is a weak (finite failure) negation meaning that according to the stored state of knowledge the statement is false.
- The conditional is designated if either the antecedent is not designated or the consequent is designated (just the standard behaviour).
- Several standard theorems are not valid: Excluded Middle, Contraposition. DeMorgan Laws do not hold for weak negation. LNC does not hold for strong negation. *Modus Tollens* does not hold. Disjunctive Syllogism holds only for weak negation.
- *Modus Ponens*, the Deduction Theorem, Contraction, Double Negation, Distribution Laws, Associativity and more all hold in **VII**.

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

QC

- Anthony Hunter and Philippe Besnard have introduced the paraconsistent propositional logic **QC**. The logic is called "QC" for "quasi-classical". **QC** does not deal with defaults and is montonic. (The intended application of **QC** is to measure the amount of inconsistent information in a database by counting non-equivalent interpretations.)
- **QC** works by taking the classical rules of natural deduction (dividing them into composition and decomposition rules) and imposing the constraint that ($\vee$I) – as the one rules that introduces new and possibly Irrelevant sentence letters – and other composition rules are to be applied only after the decomposition rules have been applied.
- **QC** by this (since no decomposition rule can be applied to an empty set of premises) has no theorems!
- Further on the derivability relation of **QC** is not transitive!

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Treating Inconsistency *vs.* Paraconsistency

- There are several approaches to deal with inconsistency in computer systems, databases or automated reasoning. Their aim usually is to show that one can still have a working system even if at some point inconsistent data crept up or are a live option.
- Such inconsistency treating approaches are, nonetheless, *not* necessary paraconsistent approaches (or applications of paraconsistency). Paraconsistency and paraconistent logics are defined [cf. Chap. 1] as avoiding trivialization in the face of actually obtaining inconsistencies (more exactly: inconsistencies obtaining in one place, this place supporting/containing both A and ¬A). If the inconsistency is *distributed* over the system or merely *implicit* without the mechanisms being available to make it explicit, this is no proper instance of the problem paraconsistency solves. If the data are rebuild to avoid the otherwise standard logic being applied to them, this again is no paraconsistent approach (of changing the logic), but a standard approach of working around inconsistencies (even if some non-standard, but non-paraconsistent logic is employed in such an approach.)

p ∧ ¬p

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Treating Inconsistency *vs.* Paraconsistency (II)

- Two examples may illustrate this distinction.
- Some weakly aggregative logics are used to model a distributed data management system that takes input from several channels, which can contradict each other [cf. (Allen 2004)]. If the knowledge operator "K" represents knowledge obtained from *some* source one may not have

  $$KA \wedge KB \supset K(A \wedge B)$$

  thus one may have K(A) and K(¬A) but not K(A ∧ ¬A). Each data stream considered for itself is taken as consistent (and explosive on non-modal contradictions), inconsistent data streams being thrown out. Non-aggregation avoids invalidating logical omniscience or Explosion, but also prevents the presence of *explicit* contradictions in the first place! So this approach is *not* paraconsistent.
- Some multi-modal epistemic logics are used for the same purpose [cf. (Calvanese et al. 2005)]. Locally inconsistent data storages/peers are thrown out and data from other data streams/peers is integrated using some non-monotonic logic *as long as consistency is preserved*. This approach thus avoids the presence of established inconsistencies and thus is *not* paraconsistent – useful as it may be otherwise.

p ∧¬p

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Complexity of Paraconsistent Logics

- Some of the paraconsistent logics we have considered are decidable, e.g. **LP** or **LFI1, J3** – and more.
- The question whether some consequence holds in these logics is of the same difficulty as in standard propositional logic (or strong modal logics like **S5**).

  $\models_{LP}$ falls in the complexity class **co-NP**.

  Proof (Outline): $\Gamma \models_{LP} A$ is true if $\Gamma \not\models_{LP} A$ is false (this being its complement problem). We know that $\Gamma \not\models_{LP} A$ if we can find an interpretation that makes all members of Γ at least true, but A false only. To do this a non-deterministic Turing-Machine "guesses" an interpretation of $\Gamma \cup \{A\}$ and checks whether it has the property in question. This can be done in a time polynomial in the length of (a great conjunction) $\Gamma \cup \{A\}$, since truth-functional de-composition effectively terminates for all **LP**-formula. Thus the complement problem of $\Gamma \models_{LP} A$ is in **NP**. So the problem is in **co-NP**. ■
- In fact are these problems **co-NP** *hard*, since standard satisfiability (SAT) can be reduced to non-entailment of some contradiction.

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Complexity of Paraconsistent Logics (II)

- In case that we consider only formulas in CNF (conjunctive normal form) the decision whether $\models_{LP}$ can be found even in polynomial time, since by the invalidity of the Disjunctive Syllogism in **LP** a formula A can be derived from a long CNF of the premises Γ only if A is subsumed by a clause of this long CNF formula expressing Γ. Thus for CNF-formula $\models_{LP}$ is in the complexity class **P**!
  (Note that this *advantage* over standard **PC** is available only because many standard consequence fail in **LP** and no new detachable conditional has been introduced.)
- Some of the paraconsistent logics we have considered are non-monotonic (i.e. adaptive). One such logic is $\mathbf{LP_m}$ (the Adaptive Logic that takes **LP** as the lower limit logic [cf. Chap. 7]).
- The decision problem for this logic $\mathbf{LP_m}$ is much harder than that of a monotonic propositional logic like **LP** itself.
  $\models_{LPm}$ falls in the complexity class Π_2.
  i.e. co- Σ_2, i.e. $\mathbf{co\text{-}NP^{NP}}$. That means: the non-deterministic Turing-Machine that solves the complementary problem $\models\!/\!=_{LPm}$ takes recourse to an oracle that is itself in **NP**.

Questions

- (Q1) Why does the constraint defining **QC** block the standard derivation of *ex contradictione qoudlibet*?
- (Q2) Apart from algebraic considerations, what may be a reason to have in **VII** for each truth value X a *simply X* and a *by default X* version, but not for the value of being contradictory?
- (Q3) In case you are familiar with *normalization* as applied in typical (SQL) databases: Why does a violation of first and second degree normalization (of *primary* and *foreign keys*) come down to inconsistent data?
- (Q4) Why can negation as *finite failure* never lead to *ex contradictione qoudlibet*?

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Exercises

- (Ex1) In case you are familiar with PROLOG try to implement the typical "penguin Tweety is a bird" example so that in one version non-monotonicity is shown and in another version by use of a defined `non_fly(X)` predicate inconsistency occurs.

- (Ex2) Outline the algorithm to check CNF-validity in **LP**.

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

$p \wedge \neg p$

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Further Reading

- On databases in general focussed to a typical commercial product cf. Soukup, Ron/Delaney, Kalen. *Inside Microsoft SQL Server*. 1999.
- On key concepts like *resolution* and *unification* see (Ben-Ari 1993).
- On PROLOG in general and negation as finite failure see Clocksin, W./Mellish, C. *Programming in Prolog*. Berlin et al., 3rd Ed. 1987.
- On paraconsistency as meta-theory see: (Damásio/Pereira 1998) and (Damásio 1996).
- On inconsistency tolerance see: Bertossi, L./Hunter, A./Schaub, T. (Eds.) *Inconsistency Tolerance*. Berlin, 2004.
- On **QC** cf. Besnard, Philippe/Hunter, Anthony. "Quasi-classical Logic.", in: *Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, Lecture Notes in Artificial Intelligence, 946, Berlin et al., 1995, pp. 44-51; Hunter's "Paraconsistent Logics", in: (Gabbay/Smets 1998); and: Wong, Paul/Besnard, Philippe. "Paraconsistent Reasoning as an Analytic Tool", *Logic Journal of the IGPL*, 9 (2001), pp. 217-29.
- On **VII** see: Ginsberg, M. "Multivalued Logics. A Uniform Approach to Inference in Artificial Intelligence", *Computer Intelligence*, 4 (1988), pp. 265-316.

Further Reading (II)

- On the complexity of some paraconsistent logics: Coste-Marquis, Sylvie/Marquis, Pierre, "On the Complexity of Paraconsistent Inference Relations", in: *Inconsistency Tolerance* (see above), and the sources mentioned in this article.

p ∧ ¬p

Manuel Bremer
Centre for Logic, Language and Information

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF