SIC: An Interactive Tool for the Design of Integrity Constraints (System Description)

François Bry, Norbert Eisinger, Heribert Schütz, Sunna Torge

in EDBT'98, Valencia, Demo Session Proceedings
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1997-18, Dezember 1997

SIC: An Interactive Tool for the Design of Integrity Constraints (System Description)

François Bry Norbert Eisinger Heribert Schütz Sunna Torge[1]

1 Introduction

Integrity constraints are declarative expressions that serve to distinguish valid database states, corresponding to possible (so-called legal) states of the real world, from invalid ones [6, 1, 8]. A valid database state is one that satisfies all integrity constraints. An important database design issue is that of "constraint satisfiability", i.e., whether the integrity constraints can be satisfied by some database states at all. Constraint satisfiability is important, since integrity constraints are in general designed before the database is populated, i.e., before any database state exists. If the integrity constraints are inconsistent or if they are not "finitely satisfiable", i.e., if they cannot be satisfied by some finite database states, then they must be considered as incorrectly designed. Note that constraint satisfiability is not the same as constraint satisfaction.

SIC is an interactive prototype to assist in the design of finitely Satisfiable Integrity Constraints. SIC combines two systems, a reasoning component and an interactive visual interface. The reasoning component is a model generator called FINFIMO [3] which extends the model generator SATCHMO [11, 4] originally developed for this application [2].

2 Model Generation with SATCHMO

Consider a company database with two relations $\texttt{employee}(\texttt{emp\_name})$ and $\texttt{assigned}(\texttt{emp\_name}, \texttt{proj\_name})$ and two integrity constraints stipulating that (1) every employee be assigned to at least one project, and (2) there must be at least one employee:

$$\forall E\,(\texttt{employee}(E) \;\rightarrow\; \exists P\,\texttt{assigned}(E,P))$$
$$\exists E\,\texttt{employee}(E)$$

SATCHMO operates on the Skolemized form

$$\forall E\,(\texttt{employee}(E) \;\rightarrow\; \texttt{assigned}(E,p(E)))$$
$$\texttt{employee}(e_0)$$

of these constraints (with a Skolem function p and Skolem constant e_0 for the existentially quantified variables) and generates the *model* (i.e., the legal database state) $\{\texttt{employee}(e_0), \texttt{assigned}(e_0,p(e_0))\}$, thus proving that the integrity constraints above are in fact satisfiable.

3 Finite Model Generation

Now consider the additional relation $\texttt{project}(\texttt{proj\_name})$ and the constraint that to every project at least one employee must be assigned. Furthermore we have the referential integrity constraints that every employee and project mentioned in the assigned relation must be listed in the employee or project relation, respectively:

$$\forall P\,(\texttt{project}(P) \;\rightarrow\; \exists E\,\texttt{assigned}(E,P))$$
$$\forall E\,\forall P\,(\texttt{assigned}(E,P) \;\rightarrow\; \texttt{employee}(E))$$
$$\forall E\,\forall P\,(\texttt{assigned}(E,P) \;\rightarrow\; \texttt{project}(P))$$

The Skolemized form of these constraints is

$$\forall P\,(\texttt{project}(P) \;\rightarrow\; \texttt{assigned}(e(P),P))$$
$$\forall E\,\forall P\,(\texttt{assigned}(E,P) \;\rightarrow\; \texttt{employee}(E))$$
$$\forall E\,\forall P\,(\texttt{assigned}(E,P) \;\rightarrow\; \texttt{project}(P))$$

Applied to the five Skolemized constraints, SATCHMO does not terminate and generates an infinite model:

$$\{\texttt{employee}(e_0), \texttt{assigned}(e_0,p(e_0)), \texttt{project}(p(e_0)), \texttt{assigned}(e(p(e_0)),p(e_0)), \texttt{employee}(e(p(e_0))), \texttt{assigned}(e(p(e_0)),p(e(p(e_0)))), \texttt{project}(p(e(p(e_0)))), \ldots\}.$$

This problem typically arises with cyclic referential integrity constraints. A first improvement of FINFIMO over SATCHMO is that it does not require integrity constraints to be expressed in the rather unnatural Skolemized form and that it can construct finite models in such cases.

Another important class of integrity constraints is the class of functional dependencies. For example, no more than one salary should be assigned to an employee:

$$\forall E\,\forall S\,\forall S'\,(\texttt{salary}(E,S) \wedge \texttt{salary}(E,S') \;\rightarrow\; S = S')$$

[1]Ludwig-Maximilians-Universität München, Institut für Informatik, Oettingenstr. 67, D-80538 München, Germany, {bry,eisinger,hschuetz,torge}@informatik.uni-muenchen.de, http://www.pms.informatik.uni-muenchen.de/

Here a model generator should not generate its own interpretation of the equality predicate, but must adhere to a predefined interpretation. FINFIMO, unlike SATCHMO, is capable of this.

4 The Reasoning Component: FINFIMO

SIC's reasoning component consists of the model generator FINFIMO. Applied to a set of integrity constraints, FINFIMO systematically tries to build models of this set. If no models can be found, i.e., if the set of integrity constraints is inconsistent, FINFIMO reports failure in finite time. If the set of integrity constraints admits some finite models, FINFIMO can generate each minimal finite model in finite time. (A model is *minimal* if no database entries can be removed without violating an integrity constraint.) If, however, all possible models of the integrity constraints are infinite, FINFIMO will undertake the construction of one such model – and, of course, never terminate. Displaying the tentative models under construction appears to give sufficient hints for database designers to recognize ill-designed integrity constraints whose models are all infinite.

Thus, FINFIMO gives rise to detect both the (undesirable) inconsistent sets of integrity constraints as well as the (desirable) finitely satisfiable sets of integrity constraints, i.e., FINFIMO is sound and complete for both finite minimal models and refutation [3]. FINFIMO of course does not detect the (undesirable) sets of integrity constraints that admit only infinite models: this property is not semi-decidable.

FINFIMO is implemented in Prolog.

5 The Visualisation Component: SNARKS

The interactive visualisation component SNARKS [9] allows both to visualize and to control the expansion of the tentative models constructed by model generators. SNARKS is generic in the sense that it can be used for the development and debugging of model generators, for designing declarative specifications, for comparing inference engines, or for illustrating the model generation process. SIC makes use of this last facility.

FINFIMO-like model generators operate on declarative specifications in a forward reasoning manner and process disjunctions by introducing ramifications into the derivations. The resulting derivations have the form of trees whose nodes are sets of formulae. SNARKS offers a wide range of operations for the interactive construction, modification, display, rearrangement, and presentation of such a tree. There are means to analyse the dependencies between derived formulae by highlighting automatically formulae that contribute to the derivation of a selected one, and, conversely, formulae that were derived using the selected one. One of the uses of this feature is to identify the reasons for contradictions in cases where expected models cannot be derived. Another debugging feature allows the interactive modification of tentative models at any point during the model generation process. Thus it is possible to anticipate the effects of corrections of the integrity constraints without reperforming the model generation process from scratch. This helps to fix former design decisions with undesirable consequences.

SNARKS consists of a Java front end for the graphics and a Prolog back end for the reasoning tasks. SIC can be called from any WWW browser supporting Java.

6 Perspectives

The reasoning component could be improved by enhancing the model generator FINFIMO into a generator of minimal models only [4]. Even though the class of integrity constraints that FINFIMO can cope with has the expressive power of first-order predicate logic without function symbols (except for constants), it is necessary to adapt the prototype SIC to a more convenient specification language, e.g., to an object-oriented language such as F-logic [10], Chimera [5] or DEL [7].

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] F. Bry, H. Decker, and R. Manthey. A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases. In *Proc. Int. Conf. Extending Data Base Technology (EDBT)*, pages 488–505, Venice, Italy, March 1988. Springer LNCS 303.

[3] F. Bry and S. Torge. Model generation for applications – a tableaux method complete for finite satisfiability. Research report PMS-FB-1997-15, Institut für Informatik,

Ludwig-Maximilians-Universität München, 1997.

[4] F. Bry and A. Yahya. Minimal model generation with positive unit hyper-resolution tableaux. In *5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 143–159. Springer LNAI 1071, 1996.

[5] S. Ceri and R. Manthey. Chimera: A model and language for active dood systems. In *Proceedings of the 2nd East-West Database Workshop*, Workshops in Computing, pages 3–16, Klagenfurt, Austria, 1993. Springer, 1995.

[6] C. Date. *An Introduction to Database Systems.* Addison-Wesley, sixth edition, 1995.

[7] O. Friesen, G. Gauthier-Villars, A. Lefebvre, and L. Vieille. Applications of deductive object-oriented databases using del. In *Workshop on Programming with Logic Databases, ILPS*, pages 1–22, 1993.

[8] S. Jajodia, W. List, G. McGregor, and L. Strous, editors. *Integrity and Internal Control in Information Systems – Volume 1: Increasing the confidence in information systems – IFIP TC-11 WG11.5 First Working Conference on Integrity and Internal Control in Information Systems*, Zürich, Switzerland, 4–5 December 1997. Chapman & Hall.

[9] M. Kettner and N. Eisinger. The tableau browser SNARKS (system description). In *14th Int. Conf. on Automated Deduction (CADE)*, pages 408–411. Springer LNAI 1249, 1997.

[10] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *JACM*, 1995.

[11] R. Manthey and F. Bry. SATCHMO: A theorem prover implemented in Prolog. In *9th Int. Conf. on Automated Deduction (CADE)*, pages 415–434. Springer LNCS 310, 1988.