Choice Logic Programs and Nash Equilibria in Strategic Games

Marina De Vos*[1] and Dirk Vermeir[1]

Dept. of Computer Science
Free University of Brussels, VUB
Pleinlaan 2, Brussels 1050, Belgium
Tel: +32 2 6293308
Fax: +32 2 6293525
{marinadv,dvermeir}@tinf1.vub.ac.be
http://tinf1.vub.ac.be

Abstract. We define choice logic programs as negation-free datalog programs that allow rules to have exclusive-only disjunctions in the head. We show that choice programs are equivalent to semi-negative datalog programs, at least as far as stable models are concerned. We also discuss an application where strategic games can be naturally formulated as choice programs; it turns out that the stable models of such programs capture exactly the set of Nash equilibria.
Keywords: nondeterminism, choice, logic programs, stable model semantics, game theory

1 Introduction

Stable model semantics[2] can be regarded as introducing nondeterminism into logic programs, as has been convincingly argued in [11, 9]. E.g. a program such as

$$p \leftarrow \neg q$$
$$q \leftarrow \neg p$$

has no (unique) total well-founded model but it has two total stable models, namely $\{p, \neg q\}$ and $\{\neg p, q\}$, representing a *choice* between p and q. This nondeterminism may not show up in the actual models, as in the program

$$p \leftarrow \neg q$$
$$q \leftarrow \neg p$$
$$p \leftarrow \neg p$$

where only the choice $\{p, \neg q\}$ turns out to be acceptable (the alternative leading to a contradiction).

In this paper, we simplify matters by providing for explicit choice sets in the head of a rule. Using $p \oplus q$ to denote a choice between p and q, the first example above can

* Wishes to thank the FWO for their support.

be rewritten as[1].

$$p \oplus q \leftarrow$$

Intuitively, $\oplus$ is interpreted as "exclusive or", i.e. either p or q, but not both, should be accepted in the above program.

It turns out that such *choice programs*, which do not use negation in the body, can meaningfully simulate arbitrary semi-negative logic programs, at least as far as their (total) stable model semantics are concerned. Since also the converse holds, we can conclude that, in a sense, choice is equivalent to negation.

Providing explicit choice as the conclusion of a rule allows for the natural expression of several interesting problems. In this paper, we show e.g. that strategic games[6] can be conveniently represented using choice programs. Moreover, the stable models of such a program characterize exactly the pure Nash equilibria of the game.

2 Choice Logic Programs

In this paper, we identify a program with its grounded version, i.e. the set of all ground instances of its clauses. This keeps the program finite as we do not allow function symbols (i.e. we stick to datalog).

Definition 1. *A **choice logic program** is a finite set of rules of the form $A \leftarrow B$ where A and B are finite sets of atoms.*

Intuitively, atoms in A are assumed to be xor'ed together while B is read as a conjunction. In examples, we often use $\oplus$ to denote exclusive or, while "," is used to denote conjunction.

Example 1 (Prisoner's Dilemma). The following simple choice logic program models the well-known prisoner's dilemma where d_i means "player i does not confess" and c_i stands for "player i confesses".

$$\begin{aligned} d_1 \oplus c_1 &\leftarrow \\ d_2 \oplus c_2 &\leftarrow \\ c_1 &\leftarrow d_2 \\ c_1 &\leftarrow c_2 \\ c_2 &\leftarrow d_1 \\ c_2 &\leftarrow c_1 \end{aligned}$$

The semantics of choice logic programs can be defined very simply.

Definition 2. *Let P be an choice logic program. The **Herbrand base** of P, denoted $\mathcal{B}_P$, is the set of all atoms occurring in the rules of P. An **interpretation** is any subset of $\mathcal{B}_P$. An interpretation I is a **model** of P if for every rule $A \leftarrow B$, $B \subseteq I$ implies that $I \cap A$ is a singleton. A model of P which is minimal (according to set inclusion) is called **stable**.*

Example 2 (Graph 3-colorability). Given the graph depicted in Fig. 2 assign each node one of three-colors such that no two adjacent nodes have the same color.

[1] Also the second example can be turned into a negation-free "choice" program, see theorem 2 below.

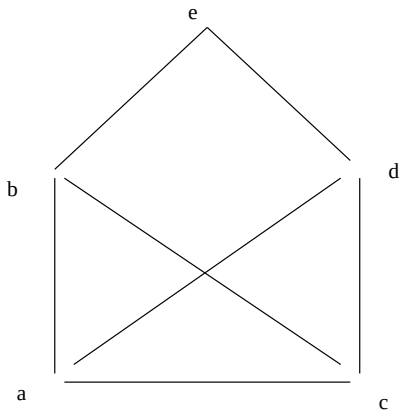

Fig. 1. Example graph which is 3-colorable.

This problem is know as graph 3-colorability and can be easily transformed in the following choice program:

$$
\begin{aligned}
color(N,b) \oplus color(N,y) \oplus color(N,r) &\leftarrow node(N) \\
&\leftarrow edge(N,J), color(N,C), color(J,C) \\
node(a) &\leftarrow \\
node(b) &\leftarrow \\
node(c) &\leftarrow \\
node(d) &\leftarrow \\
node(e) &\leftarrow \\
edge(a,b) &\leftarrow \\
edge(a,c) &\leftarrow \\
edge(a,d) &\leftarrow \\
edge(b,c) &\leftarrow \\
edge(b,e) &\leftarrow \\
edge(c,d) &\leftarrow \\
edge(d,e) &\leftarrow
\end{aligned}
$$

The first rule states that every node should take one and only one of the three available colors: black (b), yellow (y) or red (r). The second demands that two adjacent nodes have different colors. All the other rules are facts describing the depicted graph.

The stable models for this program reflect the possible solutions for this graph's 3-colorability:

$$
\begin{aligned}
N_1 &= F \cup \{color(a,b), color(b,r), color(c,y), color(d,r), color(e,b)\} \\
N_2 &= F \cup \{color(a,b), color(b,r), color(c,y), color(d,r), color(e,y)\} \\
N_3 &= F \cup \{color(a,b), color(b,y), color(c,y), color(d,y), color(e,b)\} \\
N_4 &= F \cup \{color(a,b), color(b,y), color(c,r), color(d,y), color(e,r)\}
\end{aligned}
$$

$$
\begin{aligned}
N_5 &= F \cup \{color(a,r), color(b,y), color(c,b), color(d,y), color(e,b)\} \\
N_6 &= F \cup \{color(a,r), color(b,b), color(c,y), color(d,b), color(e,y)\} \\
N_7 &= F \cup \{color(a,r), color(b,b), color(c,y), color(d,b), color(e,r)\} \\
N_8 &= F \cup \{color(a,r), color(b,y), color(c,b), color(d,y), color(e,r)\} \\
N_9 &= F \cup \{color(a,y), color(b,r), color(c,b), color(d,r), color(e,b)\} \\
N_{10} &= F \cup \{color(a,y), color(b,r), color(c,b), color(d,r), color(e,b)\} \\
N_{11} &= F \cup \{color(a,y), color(b,b), color(c,r), color(d,b), color(e,r)\} \\
N_{12} &= F \cup \{color(a,y), color(b,b), color(c,r), color(d,b), color(e,y)\}
\end{aligned}
$$

where F stands for the sets of facts from the program.

It turns out that choice logic programs can simulate semi-negative datalog programs, using the following transformation, which resembles the one used in [10] for the transformation of general disjunctive programs into negation-free disjunctive programs.

Definition 3. *Let P be a semi-negative logic program. The corresponding choice logic program $P_\oplus$ can be obtained from P by replacing each rule $r : a \leftarrow B, \neg C$ from P with $B \cup C \subseteq \mathcal{B}_P$ and $C \neq \emptyset$, by*

$$
\begin{aligned}
a_r \oplus K_C \leftarrow B \quad (r'_1) \\
a \leftarrow a_r \quad (r'_2) \\
\forall c \in C \quad \cdot K_C \leftarrow c \quad (r'_3)
\end{aligned}
$$

where a_r and K_C are new atoms that are uniquely associated with the rule r.

Intuitively, K_C is an "epistemic" atom which stands for "the (non-exclusive) disjunction of atoms from C is believed". If the positive part of a rule in the original program P is true, $P_\oplus$ will choose (rules r'_1) between accepting the conclusion and K_C where C is the negative part of the body; the latter preventing rule application. Each conclusion is tagged with the corresponding rule (r'_2), so that rules for the same conclusion can be processed independently. Finally, the truth of any member of C implies the truth of K_C (rules r'_3).

Definition 4. *Let P be a semi-negative logic program and let $P_\oplus$ be the corresponding choice logic program. An interpretation I for $P_\oplus$ is called **rational** iff:*

$$\forall K_C \in I \cdot I \cap C \neq \emptyset$$

Intuitively, a rational interpretation contains a justification for every accepted K_C.

Theorem 1. *Let P be a semi-negative datalog program. M is a rational stable model of $P_\oplus$ iff $M \cap \mathcal{B}_P$ is a (total) stable model of P.*

The rationality restriction is necessary to prevent K_C from being accepted without any of the elements of C being true. For positive-acyclic programs, we can get rid of this restriction.

Definition 5. *A semi-negative logic program P is called **positive-acyclic** iff there is an assignment of positive integers to each element of $\mathcal{B}_P$ such that the number of the head of any rule is greater than any of the numbers assigned to any non-negated atom appearing in the body.*

Note that, obviously, all stratified[8] programs are positive-acyclic. Still, many other "nondeterministic" programs such as

$$p \leftarrow \neg q$$
$$q \leftarrow \neg p$$

are also positive-acyclic.

Theorem 2. *Let P be a semi-negative positive-acyclic datalog program. There exists a choice logic program P_c such that M is a stable model of P_c iff $M \cap \mathcal{B}_P$ is a stable model of P.*

We illustrate the construction underlying theorem 2 on the following program.

$$p \leftarrow \neg q$$
$$p \leftarrow \neg p$$
$$q \leftarrow \neg p$$

The equivalent choice logic program is

$$p \oplus p^- \leftarrow$$
$$q \oplus q^- \leftarrow$$
$$p^- \leftarrow p, q$$
$$q^- \leftarrow p$$
$$p \leftarrow q^-$$
$$p \leftarrow p^-$$
$$q \leftarrow p^-$$

Intuitively, p stands for "there is a proof for p" while p^- stands for "there is no proof for p". The first two rules force the program to choose between these alternatives for every atom in the program. The rules concluding p^- (or q^-) are constructed in such a way that the truth of the body effectively blocks all possible proofs for p (resp. q). Note that the example has a single stable model $\{p, q^-\}$ which corresponds to the original's stable model $\{p, \neg q\}$.

Choice programs can be trivially simulated by semi-negative datalog programs.

Theorem 3. *Let $P_\oplus$ be a choice program. There exists a semi-negative datalog program P such that M is a stable model of $P_\oplus$ iff M is a stable model of P.*

3 Computing Stable Models

Stable models for choice logic programs can be computed by a simple "backtracking fixpoint" procedure. Essentially, one extends an interpretation by applying an immediate consequence operation, then makes a choice for every applicable rule[2] which is not

[2] A rule $A \leftarrow B$ is applicable w.r.t. an interpretation I iff $B \subseteq I$.

actually applied[3], backtracking if this leads to an inconsistency (i.e. the current interpretation cannot be extended to a model).

```
function fix(set<atom> N): set<atom>
{
set<atom> M = ∅;
repeat
        for each (A ← B) ∈ P do
                if B ⊆ M ∧ ∃a ∈ A · a ∉ M ∪ N ∧ (A \ {a}) ⊆ N then
                        M = M ∪ {a}
until no change in M
return M
}
```

Fig. 2. *fix* is an auxiliary function for *BF*.

Figure 3 on page 7 presents such a fixpoint computation procedure *BF* which is called by the main program using

$$BF(fix(\emptyset),\emptyset).$$

We believe this procedure to be simpler than a similar one presented in [11] for semi-negative logic programs. *BF* uses an auxiliary function *fix* depicted in Fig. 2 on page 6. *Fix* is a variation on the immediate consequence operator: it computes the least fixpoint of this operator given a fixed set N of atoms that are considered to be false. Note that *fix* is deterministic since it only draws tentative conclusions from an applicable rule if there is but one possible choice for an atom in the head that will be true.

The main procedure *BF* in Fig. 3 takes two sets of atoms, M and N, containing the atoms that already have been determined to be *true* and *false*, respectively. Note that, because $M = fix(N)$ upon entry, there are no applicable rules in P that have but one undefined atom in the head. The procedure *BF* works by first verifying that no rules are violated w.r.t. the current M and N (see the definition of V in Fig. 3). It then computes the set C of applicable (but unapplied) rules for which a choice can be made as to the atom from the head that needs to be true in order to apply the rule. If there are no such rules, we have a model. Otherwise, the algorithm successively selects a rule r from C and a possible choice c from the head of r that would make it applied. This choice is "propagated" using *fix*, after which *BF* is called recursively using the new versions of M and N[4].

Theorem 4. *Let P be a choice logic program. Then BF(fix(∅),∅), where BF is described in Fig. 3 terminates and computes exactly the set of stable models of P.*

Note that, because of theorem 1, *BF* can be easily modified to compute the stable models of any semi-negative logic program through its equivalent choice logic program.

[3] An applicable (w.r.t. an interpretation I) rule is applied iff $A \cap I$ is a singleton.

[4] Clearly, the algorithm can be made more efficient, e.g. by memoizing more intermediate results.

```
procedure BF(M, N :set<atom>)
{
set<rule> V = {(A ← B) ∈ P | B ⊆ M ∧ (#(A ∩ M) > 1 ∨ A ⊆ N)}
if (V ≠ ∅)
        return /* because some rules are violated */
set<rule> C = {(A ← B) ∈ P | B ⊆ M ∧ #(A \ (M ∪ N)) > 1}
if (C = ∅)
        output M
else
        for each ((A ← B) ∈ C) {
                set<atom> c = A \ (M ∪ N)
                for each (a ∈ C) {
                        N = N ∪ (c \ {a})
                        BF(fix(N),N)
                        }
                }
}
```

Fig. 3. The *BF* (backtracking fixpoint) procedure for Choice Logic Programs.

	$Bach$	$Stravinsky$
$Bach$	$2, 1$	$0, 0$
$Stravinsky$	$0, 0$	$1, 2$

Fig. 4. Bach or Stravinsky (BoS)

4 An Application to Strategic Games

A strategic game models a situation where several agents (called players) independently make a decision about which action to take, out of a limited set of possibilities. The result of the actions is determined by the combined effect of the choices made by each player. Players have a preference for certain outcomes over others. Often, preferences are modeled indirectly using the concept of *payoff* where players are assumed to prefer outcomes where they receive a higher payoff.

Example 3 (Bach or Stravinsky). Two people wish to go out together to a music concert. They can choose for Bach in one theater or for Stravinsky in another one. Their main concern is to be together, but one person prefers Bach and the other prefers Stravinsky. If they both choose Bach then the person who preferred Bach gets a payoff of 2 and the other one a payoff of 1. If both go for Stravinsky, it is the other way around. If they pick different concerts, they both get a payoff of zero.
The game is represented in Fig. 4. One player's actions are identified with the rows and the other player's with the columns. The two numbers in the box formed by row r and column c are the players' payoffs when the row player chooses r and the column player chooses c, the first component being the payoff of the row player.

Definition 6 ([6]). *A **strategic game** is a tuple $\langle N, (A_i)_{i \in N}, (\succeq_i)_{i \in N} \rangle$ where*

- *N is a finite set of **players**;*

– *for each player $i \in N$, A_i is a nonempty set of **actions** that are available to her (we assume that $A_i \cap A_j = \emptyset$ whenever $i \neq j$) and,*
– *for each player $i \in N$, $\geq_i$ is a **preference relation** on $A = \times_{j \in N} A_j$*

*An element $\mathbf{a} \in A$ is called a **profile**. For a profile $\mathbf{a}$ we use $\mathbf{a}_i$ to denote the component of $\mathbf{a}$ in A_i. For any player $i \in N$, we define $A_{-i} = \times_{j \in N \setminus \{i\}} A_j$. Similarly, an element of A_{-i} will often be denoted as $\mathbf{a}_{-i}$. For $\mathbf{a}_{-i} \in A_{-i}$ and $a_i \in A_i$ we will abbreviate as $(\mathbf{a}_{-i}, a_i)$ the profile $\mathbf{a}' \in A$ which is such that $\mathbf{a}'_i = a_i$ and $\mathbf{a}'_j = \mathbf{a}_j$ for all $j \neq i$.*

Playing a game $\langle N, (A_i)_{i \in N}, (\geq_i)_{i \in N} \rangle$ consists of each player $i \in N$ selecting a single action from the set of actions A_i available to her. Since players are thought to be rational, it is assumed that a player will select an action that leads to a "preferred" profile. The problem, of course, is that a player needs to make a decision not knowing what the other players will choose.

The notion of Nash equilibrium shows that, in many cases, it is still possible to limit the possible outcomes (profiles) of the game.

Definition 7. *A **Nash equilibrium** of a strategic game $\langle N, (A_i)_{i \in N}, (\geq_i)_{i \in N} \rangle$ is a profile $\mathbf{a}^*$ satisfying*

$$\forall a_i \in A_i \cdot (\mathbf{a}^*_{-i}, \mathbf{a}^*_i) \geq_i (\mathbf{a}^*_{-i}, a_i)$$

Intuitively, a profile $\mathbf{a}^*$ is a Nash equilibrium if no player can unilaterally improve upon his choice. Put in another way, given the other players' actions $\mathbf{a}^*_{-i}$, $\mathbf{a}^*_i$ is the best player i can do[5].

Given a strategic game, it is natural to consider those moves that are best for player i, given the other players' choices.

Definition 8. *Let $\langle N, (A_i)_{i \in N}, (\geq_i)_{i \in N} \rangle$ be a strategic game. The **best response** function B_i for player $i \in N$ is defined by*

$$B_i(\mathbf{a}_{-i}) = \{a_i \in A_i \mid \forall a'_i \in A_i \cdot (\mathbf{a}_{-i}, a_i) \geq_i (\mathbf{a}_{-i}, a'_i)\}$$

The following definition shows how games allow an intuitive representation as choice logic programs.

Definition 9. *Let $G = \langle N, (A_i)_{i \in N}, (\geq_i)_{i \in N} \rangle$ be a strategic game. The choice logic program P_G associated with G contains the following rules:*

– *For each player i, P_G contains the rule $A_i \leftarrow$. This rule ensures that each player i chooses exactly one action from A_i.*
– *For each player i and for each profile $\mathbf{a} \in A_{-i}$, P_G contains a rule[6] $B_i(\mathbf{a}) \leftarrow \mathbf{a}$. It models the fact that a player will select a "best response", given the other players' choices.*

Essentially, P_G simply forces players to choose an action. Moreover, the action chosen should be a "best response" to the other players' actual choices.

[5] Note that the actions of the other players are not actually known to i.
[6] We abuse notation by writing $\mathbf{a}$ for the set of components of $\mathbf{a}$.

Theorem 5. *For every strategic game $G = \langle N, (A_i)_{i \in N}, (\geq_i)_{i \in N} \rangle$ there exists a choice logic program P_G such that the set of stable models of P_G coincides with the set of Nash equilibria of G.*

Example 4. Let us reconsider the Bach or Stravinsky game of example 3. This game has two Nash equilibria, namely:

$$(Bach_1, Bach_2) \text{ and } (Stravinsky_1, Stravinsky_2)$$

The corresponding choice logic program is:

$$\begin{aligned} b_1 \oplus s_1 &\leftarrow \\ b_2 \oplus s_2 &\leftarrow \\ b_1 &\leftarrow b_2 \\ s_1 &\leftarrow s_2 \\ b_2 &\leftarrow b_1 \\ s_2 &\leftarrow s_1 \end{aligned}$$

where b_i and s_i are shorthands for player i choosing respectively $Bach$ or $Stravinsky$. This program has two stable models, namely $\{s_1, s_2\}$ and $\{b_1, b_2\}$ that correspond to the Nash equilibria of the game.

Example 5. The program in example 1 is the choice logic program corresponding to the strategic game depicted in Fig. 5. Here two prisoners are interrogated in separate

	Do not confess	*Confess*
Do not confess	$3, 3$	$0, 4$
Confess	$4, 0$	$1, 1$

Fig. 5. Prisoner's Dilemma

rooms. Each one must decide whether or not to confess. Confessing implicates the other prisoner and may result in a lighter sentence, provided the other prisoner did not confess. This game has one Nash equilibrium

$$\{Confess_1, Confess_2\}$$

corresponding the single stable model of the program of example 1.

Note that the construction of P_G can be regarded as an encoding of the fact that the rationality and preferences of the players are common knowledge, as all rules interact and "cooperate" to verify atoms. This observation opens the possibility of extending the present approach to one where players may not be fully aware of each other's beliefs. This could be done, e.g. by considering a "choice" variation of "ordered logic programs"[3–5].

Another interesting aspect of theorem 5 is that, in combination with theorem 4, it provides a systematic method for the computation of Nash equilibria for (finite) strategic games.

Corollary 1. *For every strategic game* $G = \langle N, (A_i)_{i \in N}, (\succeq_i)_{i \in N} \rangle$ *there exists a semi-negative datalog program program* P_G *such that the set of stable models of* P_G *coincides with the set of Nash equilibria of* G.

5 Relationship to Other Approaches and Directions for Further Research

The logical foundations of game theory have been studied for a long time in the confines of epistemic logic, see e.g. [1] for a good overview. However, to the best of our knowledge, very little has been done on using logic programming-like formalisms to model game-theoretic concepts.

An important exception is [7] which introduces a formalism called "Independent Choice Logic" (ICL) which uses (acyclic) logic programs to deterministically model the *consequences* of choices made by agents. Since choices are external to the logic program, [7] restricts the programs further to not only be deterministic (i.e. each choice leads to a unique stable model) but also independent in the sense that literals representing alternatives may not influence each other, e.g. they may not appear in the head of rules. ICL is further extended to reconstruct much of classical game theory and other related fields.

The main difference with our approach is that we do not go outside of the realm of logic programming to recover the notion of Nash equilibria. Contrary to ICL, we *rely* on nondeterminism to represent alternatives, and on the properties of stable semantics to obtain Nash equilibria. As for the consequences of choices, these are represented in choice logic programs, much as they would be in ICL.

The present paper succeeded in recovering Nash equilibria without adding any fundamentally new features to logic programs (on the contrary, we got rid of negation in the body). However, the results are restricted to so-called "pure" equilibria where each participant must choose a single response. We would like to extend the formalism further to cover, in a similar way, also other game-theoretic notions. E.g. we are presently working on extending our approach to represent mixed equilibria (which are probability distributions over alternatives) as well. Finally, as mentioned in Sec. 4, using (an extension of) ordered logic could simplify the introduction of epistemic features into the formalism.

References

1. Pierpaolo Battigalli and Giacomo Bonanno. Recent Results on Belief, Knowledge and the Epistemic Foundations of Game Theory. Working Paper.
2. Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, pages 1081–1086, Seattle, 1998. ALP, IEEE, The MIT Press.
3. P. Geerts and D. Vermeir. Credulous and Autoepistemic Reasoning using Ordered Logic. *Proceedings of the First International Workshop on Logic Programming and Non-Monotonic Reasoning*, page 21–36, MIT Press, 1991.

4. P. Geerts and D. Vermeir and D. Nute. Ordered logic: Defeasible Reasoning for Multiple Agents. *Decision Support Systems*, **11** (1994) 157–190.
5. E. Laenens and D. Vermeir. A Fixpoint Semantics of Ordered Logic. *Journal of Logic and Computation*, **1(2)**(1990) 159-185.
6. M. J. Osborne and A. Rubinstein. *A Course in Game Theory*, MIT Press,1994.
7. David Poole. The Independent Choice Logic for Modelling Multiple Agents under Uncertainty. *Artificial Intelligence*, **94(1–2)** (1997) 7–56.
8. T. C. Przymusinski. Every Logic Program Has a Natural Stratification and an Iterated Fixed Point Model. In *Proceedings of the Eight ACM Symposium on Principles of Database Systems*, pages 11–21. Association for Computing Machinery, 1989.
9. D. Sacca. Deterministic and Non-Deterministic Stable Models. *Logic and Computation*, **5** (1997) 555–579.
10. Chiaki Sakama and Katsumi Inoue An Alternative Approach to the Semantics of Disjunctive Logic Programs and Decductive Databases. *Journal of Automated Reasoning*, **13** (1994) 145–172.
11. D. Sacca and C. Zaniolo. Stable Models and Non-Determinism for Logic Programs with Negation. In *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 205-218. Association for Computing Machinery, 1990.