1E4-05

仮説の動的な更新における近傍探索

A Neighborhood-Search Method for Updating Hypotheses

藤原 翔一朗　　山本 章博
Shoichirou Fujiwara　Akihiro Yamamoto

京都大学 情報学研究科
Graduate School of Informatics Kyoto University

In this paper, we treat dynamic information processing in the framework of hypothetical reasoning with predicate logic. We propose a new method for updating hypotheses by following up nonmonotonic changes of background knowledge. Model theoretical approaches have mainly been employed for this subject, for example, belief revision and truth maintenance. In contrast to these previous works, we adopt proof theoretical approach because of the following two advantages: One is that we can apply the Contradiction Backtracing, which was devised in the field of Machine Learning in order to find the part of current hypotheses to be revised. The other is that we can introduce the notion of neighborhood-search and give a method of rebuilding proof based on it to find new hypotheses following up changes of background knowledge.

1. まえがき

近年携帯端末の普及などによって身近な場面でも動的な情報を扱う機会が増えるに伴い動的な情報を処理するための推論方式に関する研究は重要性を増している．本稿では，論理プログラミングに基づく仮説推論 (hypothetical reasoning) の枠組みにおいて，事実が変化せず背景知識が動的に変化する場合に仮説をそれに追従させて更新するための新しい推論手法を提案する．

この手法は，帰納論理プログラミングで考案された矛盾点追跡 (contradiction backtracing)([Shapiro 91]) を利用して仮説の導出中の矛盾点を発見し，発見した矛盾点から仮説を導出し直すという考え方を基本とする．そして，更新前の導出と似た導出を構成して新しい仮説を導出する近傍探索という概念を採用する．近傍探索という概念は我々も普段利用することが多い．例えば，車に乗ってある目的地まで向かう際，通る予定の道が何らかの理由で利用できなくなったとする．この状況では迂回路を通らねばならないが，できるだけ予定の道と近く，所要時間もあまり変わらない迂回路を我々は選ぼうとする．これは道の場所と所要時間に関する近傍探索と考えることができる．このような近傍探索の導入は，我々の日常の知的行為を自然な形でモデル化することを目指すものである．

2. 節論理による仮説の生成

本稿での仮説推論は，確定プログラム (definite program) B 及びユニットプログラム (unit program)E が与えられたとき $B \wedge \neg(E\sigma) \models \neg H$　　(σ は Skolem 代入) なるユニットプログラム H を求めることである．最初に B を固定して H を求める手法として**CS-SOLD 導出**(CS-SOLD-derivation) を提案する[*1]．P を確定プログラム，G をゴール節 (goal clause) とする．ゴール節 F を帰結とする (P,G) の CS-SOLD 導出とは，次の条件を満たす 4 つ組 $D_i = [G_i, F_i, \theta_i, C_i]$ $(i \geq 1)$ の有限列からなる導出である．

1. G_i，F_i はゴール節，θ_i は代入，C_i は P 中に存在する確定節の変種で，変数が正規分離化されているものである．
2. $G_1 = G$ かつ $F_1 = \Box$.
3. $G_n = \Box$ かつ $F_n = F$.
4. $1 \leq i \leq n-1$ であるすべての i に対して，G_i，F_i を，$G_i = \leftarrow A_1, A_2, \ldots, A_{k_i}$，$F_i = \leftarrow B_1, B_2, \ldots, B_{h_i}$，$C_i = M_0 \leftarrow M_1, M_2, \ldots, M_{l_i}$ (k_i，h_i，l_i は 0 以上の整数) と表現すると，計算規則 R によって選択された A_{m_i} $(1 \leq m_i \leq k_i)$ に対して，以下の (a)，(b)，(c) のいずれかが成立する．

(a) (Resolution) θ_i は M_0 と A_{m_i} の最汎単一化代入 (mgu, most general unifier) であり，

$$G_{i+1} = \leftarrow (A_1, A_2, \ldots, A_{m_i-1}, M_1, M_2, \ldots, M_{l_i}, A_{m_i+1}, \ldots, A_{k_i})\theta_i,$$
$$F_{i+1} = F_i\theta_i.$$

(b) (Normal Skip) $C_i = \Box$，$\theta_i = \epsilon$ (空代入) であり，

$$G_{i+1} = \leftarrow A_1, A_2, \ldots, A_{m_i-1}, A_{m_i+1}, \ldots, A_{k_i},$$
$$F_{i+1} = \leftarrow B_1, B_2, \ldots, B_{h_i}, A_{m_i}.$$

(c) (Member Skip) $C_i = \Box$，$\theta_i = \epsilon$ で，$A_{m_i} \in \{B_1, \ldots, B_{h_i}\}$ (変種も区別する) であり，

$$G_{i+1} = \leftarrow A_1, A_2, \ldots, A_{m_i-1}, A_{m_i+1}, \ldots, A_{k_i},$$
$$F_{i+1} = F_i.$$

5. 計算規則 R は，Resolution が適用できないリテラルを優先して選択する．
6. 選択されたリテラルについて，そのリテラルを出現させた Resolution 以来導出過程で Resolution が行われていない場合に限り Skip 操作を行うことができる．

ゴール節 F を帰結とする (P,G) の CS-SOLD 導出列とは，上の条件を満たす 4 つ組 $D_i = [G_i, F_i, \theta_i, C_i]$ $(i \geq 1)$ の有限列である．CS-SOLD 導出列の要素 $[G, F, \theta, C]$ のうち，G の部分を**ゴール**，F の部分を**帰結**と呼ぶ．CS は Context Sensitive の頭文字である．CS-SOLD 導出を用いて帰結を導出するような融合を**CS-SOLD 融合**(CS-SOLD-resolution) と呼ぶ．

連絡先: 山本章博，京都大学情報学研究科，〒 606-8501 京都市左京区吉田本町，Phone: 075-753-5995，Fax: 075-753-5628，Email: akihiro@i.kyoto-u.ac.jp

*1 CS-SOLD 導出は SOLD 導出 (SOLD-derivation) ([Yamamoto 00a, 山本 00b]) の応用である．

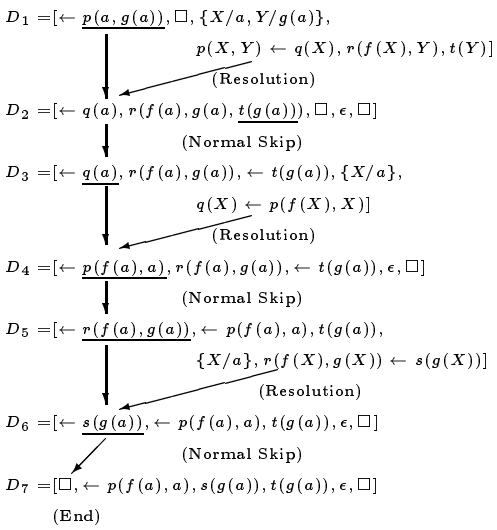

図 1: CS-SOLD 導出の例

例 1 確定プログラム P とゴール節 G をそれぞれ

$$P = \left\{ \begin{array}{l} p(X,Y) \leftarrow q(X), r(f(X), Y), t(Y), \\ q(X) \leftarrow p(f(X), X), \\ r(f(X), g(X)) \leftarrow s(g(X)) \end{array} \right\},$$

$$G = \leftarrow p(a, g(a))$$

とする．このとき，図 1 のように (P, G) の CS-SOLD 導出を行うことにより，$F = \leftarrow p(f(a), a), s(g(a)), t(g(a))$ が帰結として得られる (ゴール中の下線は R によって選択されたリテラルを指す)．

ゴール節 F が (P, G) の SOLD 導出の帰結であることと，F が (P, G) の CS-SOLD 導出の帰結であることは同値である．また，これと SOLD 融合の完全性と健全性 [Yamamoto 00a] より，CS-SOLD 融合は完全性と健全性が成立する．

3. 節論理における仮説の更新

本章では，CS-SOLD 導出を用いて B が動的に変化する場合に H をそれに追従させて更新する手法を与える．本稿において，仮説の更新問題は次のように定式化される:
$B \wedge \neg(E\sigma) \models \neg H$ (σ は Skolem 代入) であり，このとき，B が変更され B' となったときに，$B' \wedge \neg(E\sigma) \models \neg H'$ なる H' を求める．

B に対する変更には次の 3 種類がある．

1. (Add) B に確定プログラム B_A を加えて B' とする．
2. (Delete) B の部分集合である確定プログラム B_D を B から取り除いて B' とする．
3. (Update) B のうち，対象となる確定プログラム B_O を確定プログラム B_N に変更して B' とする．

ここで，以下で用いる用語，記号の定義を行っておく．(P, G_1) の CS-SOLD 導出 (列)$D_1, D_2, \ldots, D_m, D_{m+1}, \ldots, D_n$ ($D_i = [G_i, F_i, \theta_i, C_i]$, $i \geq 1$) が与えられたとき，部分列 $D_{m+1}, D_{m+2}, \ldots, D_k$ を**CS-SOLD 部分導出 (列)** といい，

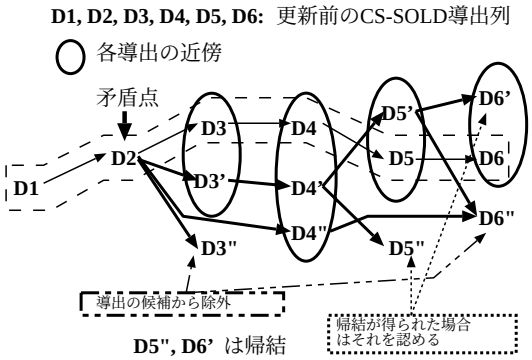

図 2: 導出の近傍を探索する

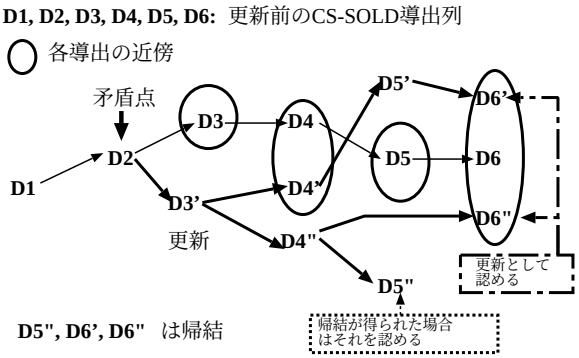

図 3: 導出の近傍を迂回する

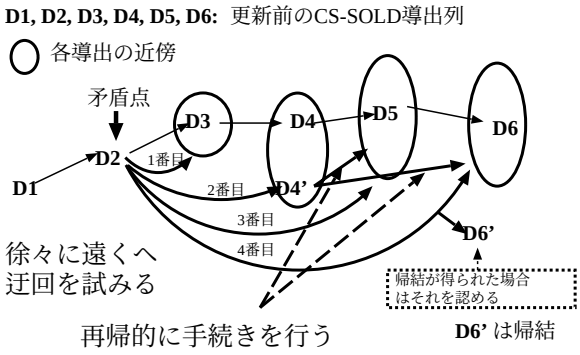

図 4: 更新手続き

この部分導出 (列) は (G_{m+1}, F_{m+1}) を**始点**，(G_k, F_k) を**終点**とするという．$G_k = \square$ すなわち $k = m$ のとき，終点を**帰結**と呼ぶ．また，$k - m$ をこの部分導出 (列) の**長さ**という．$D_1, D_2, \ldots, D_m$ をこの部分導出 (列) の**前部分導出列**という．

更新前の CS-SOLD 導出列を CS_B ，更新後の CS-SOLD 導出列を CS_A とする．仮説は帰結の否定であることに注意されたい．また特に断らない限り，CS-SOLD(部分) 導出 (列) を，単に (部分) 導出 (列) と表記する．

提案する仮説の更新手法では，導出の矛盾点を発見し，新しい背景知識の下で導出が成立するように矛盾点以後の導出をやり直すことによって仮説の更新を実現する．

(P, G) の導出 $D_1, D_2, \ldots, D_n (n \geq 1)$ と新しい確定プログラム P' が与えられたとき，導出の**矛盾点**(contradicted point) とは，$D_1, D_2, \ldots, D_n$ において，C_i $(1 \leq i \leq n)$ が $\square$ でなくかつ $C_i \notin P'$ という条件を満たす添字 i が最小の D_i である．導出の矛盾点を求めるアルゴリズムは簡単に得られる．導出列を先頭から確かめていけばよい．

矛盾点を求めたらそこから導出を更新して新たな仮説を求めるのであるが，仮説の更新という観点から考えて更新前の仮説と関連が強い仮説を求めるようにしたい．本手法では新旧の仮説の導出過程の関連性に注目する．すなわち，旧い導出と似た導出を行い，その結果得られた仮説を旧仮説と関連が強い仮説と定義する．もとの導出と似た導出を求めるために，導出列の要素に対する**近傍探索**(neighborhood-search) という考え方を導入する．

まず，任意のゴール節 E の近傍であるゴール節の集合 $Nei(E)$ が与えられているとする．導出列の要素 $D = [G, F, \theta, C]$ に対する**近傍**(neighborhood) $N(D)$ とは，次の条件を満たす集合である:

$N(D) = \{[G', F', \theta', C'] \mid G' \in Nei(G)$ かつ $F' \in Nei(F)\}$.

ただし，G'，F' はゴール節，θ' は代入，C' は確定節である．

導出の近傍探索の概要は以下の通りである．原則として導出の近傍に位置する導出列の要素だけを更新後の導出として認め，それ以外の帰結に達していない導出列の要素については探索を行わない．ただし，導出列の要素が帰結に達している場合はそれを認める (図 2)．なぜなら本手法は，先に述べたように「似た導出」を探すのであって,「似た帰結」を探すのではないからである．

上にあげた背景知識の 3 種類の変更方法に対して仮説の更新手続きを定式化する．背景知識に対する変更が Add である場合，節論理は単調論理であるので導出の矛盾点は存在しない．Update の場合，Add と Delete の組み合わせで対応する．

背景知識に対する変更が Delete である場合，導出列に矛盾点が生じる可能性がある (図 2，図 3，図 4 の D_2)．よって，近傍探索の考え方を基に仮説を更新する．前述の原則に忠実に従うと，図 2 のように新しい帰結に至る導出列中の全ての要素 (図 2 では D_5''，D_6') が更新前の導出列中の要素のいずれかの近傍 (図 2 では D_3，D_4，D_5，D_6) に位置しなければならない．しかし，それでは，導出の再構築に対する制限が強すぎる．最終目標は H' を導出することであるので，厳密に近傍だけを経由する導出が得られない場合には，近傍を必ずしも経由しない部分導出列を認める．これを**迂回**と呼ぶことにする．そしてその部分導出列を組み合わせて得た導出列を，更新後の導出列とすることも認める．図 3 では，D_5''，D_6'，D_6'' へ至る導出列を更新後の導出列として認める．また，図 4 に示すようにこのような迂回部分ができるだけ小さくなるようにする．一連の操作をアルゴリズム UpdateFromDeleted として図 5 に記す．

Algorithm *UpdateFromDeleted*　*Delete* の場合に導出を更新

Inputs CS_B : 更新前の導出列
B : 更新前の背景知識
B_D : B の部分集合

Output 更新後の導出列

Method

begin
$B' := B - B_D$;
if 矛盾点 D_{btp} が存在 ***then***
begin
$D_{btp}, D_{btp+1}, \ldots, D_n$ の近傍を経由 (一部迂回) して帰結に至り，前部分導出列が $D_1, D_2, \ldots, D_{btp-1}$ である部分導出列を探索;
if 探索に成功 ***then***
return 探索した一連の導出列
else/*探索に失敗*/
(B', G_1) の *CS-SOLD* 導出を行う;
end;
else /*矛盾点が無い*/
return CS_B;
end;

図 5: アルゴリズム UpdateFromDeleted

表 1: 背景知識 B の列車の運行に関する部分

知識	節
列車 3A 新大阪発博多行	$train(3A$, 博多, 新大阪)
列車 3A は新大阪 9:29 発	$start(3A$, 新大阪, 9:29)
列車 3A は新神戸 9:41 着	$stop(3A$, 新神戸, 9:41)
...	...
列車 3A は博多 11:57 着	$stop(3A$, 博多, 11:57)
... (他の列車についても同様)	

4. 実問題への応用

ここまでで定式化した枠組みの実問題への応用例として，列車の乗り換え問題 (以下，本問題と呼ぶ) を挙げる．ある駅 (出発駅) からある駅 (目的駅) まで列車を利用して向かうとする．出発できる時間，目的駅に到着しなければならない時間も与えられているとする．このとき，次々と変化する列車運行状況をもとに列車の乗り継ぎ経路を求める．

本問題における背景知識 B は表 1 に示す単位節と，表 2 のような確定プログラムを用いて表現される．列車の運行状況の変化 (運行中止，臨時便運行等) が B の変化で表現される．そして，旅行行程の制約に対応する事実 E は次のように与えられる．

$travel$(相生, 新山口, 7:30, 10:30).
(相生を 7:30 以降に出発して，新山口に 10:30 までに到着)

上のように B，E を定めて CS-SOLD 導出を行うことによって仮説 H，すなわち列車の乗り換え経路が得られる．述語 $early$ に関しては，適切な Herbrand instance $early(t, u)$ が単位節として暗に用意されていると仮定する．例えば図 6 のように導出が行われ H として以下のユニットプログラムが得られたとする．

$take(633A$, 相生, 広島, 7:55, 9:40),
$take(349A$, 広島, 新山口, 9:44, 10:16).

このとき，列車 349A が運行中止になったとする．背景知識の

表 2: 背景知識 B の乗り換え等に関する部分

時刻 ET は時刻 LT より早い $\Rightarrow$ $early(ET, LT)$
駅 A から駅 B まで列車 Tr に乗る (駅 A を DT 発, 駅 B に AT 着) $\Rightarrow$ $take(Tr, A, B, DT, AT)$
列車 Tr が時刻 DT 駅 A 発で時刻 AT 駅 B 着 $\Rightarrow$ $go(N, A, B, DT, AT)$
$go(N, A, B, DT, AT) \leftarrow$ $start(N, A, DT), stop(N, B, AT), early(DT, AT)$
目的駅 Des, 出発駅 Str, 到着時刻 AT, 出発時刻 DT で旅行 $\Rightarrow$ $travel(Str, Des, DT, AT)$
$travel(Str, Des, DT, AT) \leftarrow$ $go(N, Str, Des, DT1, AT1), early(DT, DT1),$ $early(AT1, AT), take(N, Str, Des, DT1, AT1)$
$travel(Str, Des, DT, AT) \leftarrow$ $go(N, Str, TmpS, DT1, TmpAT), early(DT, DT1),$ $early(TmpAT, AT), travel(TmpS, Des, TmpAT, AT)$ $take(N, Str, TmpS, DT1, TmpAT)$

(ゴールと帰結のみ記し，導出列は一部省略)

D_a =[$\leftarrow travel$(相生, 新山口, 7:30, 10:30), □],

D_b =[($\leftarrow go(633A$, 相生, 広島, 7:55, 9:40), $early$(7:30, 7:55),
$early$(9:40, 10:30), $travel$(広島, 新山口, 9:40, 10:30),
$take(633A$, 相生, 広島, 7:55, 9:40)), □],

D_c =[($\leftarrow go(633A$, 相生, 広島, 7:55, 9:40), $travel$(広島, 新山口, 9:40, 10:30)),
($\leftarrow take(633A$, 相生, 広島, 7:55, 9:40))],

D_d =[($\leftarrow start(633A$, 相生, 7:55), $stop(633A$, 広島, 9:40), $early$(7:55, 9:40)
$travel$(広島, 新山口, 9:40, 10:30)), ($\leftarrow take(633A$, 相生, 広島, 7:55, 9:40))],

D_e =[$\leftarrow travel$(広島, 新山口, 9:40, 10:30), $\leftarrow take(633A$, 相生, 広島, 7:55, 9:40)],

D_f =[($\leftarrow go(349A$, 広島, 新山口, 9:44, 10:16), $early$(9:40, 9:44),
$early$(10:16, 10:30), $take(349A$, 広島, 新山口, 9:44, 10:16)),
($\leftarrow take(633A$, 相生, 広島, 7:55, 9:40)],

D_g =[($\leftarrow go(349A$, 広島, 新山口, 9:44, 10:16)),
($\leftarrow take(633A$, 相生, 広島, 7:55, 9:40, $take(349A$, 広島, 新山口, 9:44, 10:16)))],

D_h =[($\leftarrow start(349A$, 広島, 9:44), $stop(349A$, 新山口, 10:16), $early$(9:44, 10:16)),
($\leftarrow take(633A$, 相生, 広島, 7:55, 9:40, $take(349A$, 広島, 新山口, 9:44, 10:16)))],

D_i =[□, ($\leftarrow take(633A$, 相生, 広島, 7:55, 9:40,
$take(349A$, 広島, 新山口, 9:44, 10:16)))].

図 6: 乗り継ぎ経路の導出

D'_e =[$\leftarrow travel$(広島, 新山口, 9:40, 10:30), $\leftarrow take(633A$, 相生, 広島, 7:55, 9:40)],

D'_f =[($\leftarrow go(1A$, 広島, 新山口, 9:52, 10:16), $early$(9:40, 9:52),
$early$(10:16, 10:30), $take(1A$, 広島, 新山口, 9:52, 10:16)),
($\leftarrow take(633A$, 相生, 広島, 7:55, 9:40)],

D'_g =[($\leftarrow go(1A$, 広島, 新山口, 9:52, 10:16)),
($\leftarrow take(633A$, 相生, 広島, 7:55, 9:40, $take(1A$, 広島, 新山口, 9:52, 10:16)))],

D'_h =[($\leftarrow start(1A$, 広島, 9:52), $stop(1A$, 新山口, 10:16), $early$(9:52, 10:16)),
($\leftarrow take(633A$, 相生, 広島, 7:55, 9:40, $take(1A$, 広島, 新山口, 9:52, 10:16)))],

D'_i =[□, ($\leftarrow take(633A$, 相生, 広島, 7:55, 9:40,
$take(1A$, 広島, 新山口, 9:52, 10:16)))].

図 7: 乗り継ぎ経路の更新

変更操作は Delete である．矛盾点を見つけ導出の再構築を通して乗り継ぎ経路を更新する．導出の矛盾点は D_e である．そこから例えば図 7 のように導出を再構築して新たな乗り継ぎ経路を以下のように求めることができる．
$take(633A$, 相生, 広島, 7:55, 9:40),
$take(1A$, 広島, 新山口, 9:52, 10:22).

5. おわりに

CS-SOLD 導出に関して，導出規則 Skip と Resolution の適用に優先順位を設けるという議論は以前から行われていた．Inoue は，ゴールから選択されたリテラルが持つ特性に応じてリテラルに適用する操作を制限するという推論方法を提案した ([Inoue 92])．CS-SOLD 融合がこの方法や SOLD 融合と異なる点は，導出規則および導出規則を適用するリテラルの選び方が，それまでの導出列全体と Resolution に用いられる確定プログラムに依存する点である．この導出の各段階に対する依存関係を導入することによって，情報の時系列的な依存関係を考慮した問題のモデル化が容易になる．

本稿の枠組みを実問題に応用するための鍵は，ゴール節間における適切な近傍の基準を用いることである．基準の一つとして，ゴール節を原子文の集合と捉え，[Hutchinson 97, Ramon 98] などで定式化されている原子文の集合に対する距離を用いることが考えられる．しかし，それがゴール節の構造に対し適切であるかどうかは今後検討する必要がある．

参考文献

[Hutchinson 97] Hutchinson, A.: Metrics on Terms and Clauses, in *Proceedings of the 9th European Conference on Machine Learning, Lecture Notes in Artificial Intelligence* 1224, pp. 138–145, Springer-Verlag (1997)

[Inoue 92] Inoue, K.: Linear Resolution for Consequence Finding, *Artificial Intelligence* 56, pp. 301–353 (1992)

[Ramon 98] Ramon, J. and Bruynooghe, M.: A Framework for Defining Distances Between First-Order Logic Objects, in Page, D. ed., *Proceedings of the 8th International Conference on Inductive Logic Programing, Lecture Notes in Artificial Intelligence* 1446, pp. 271–280, Springer-Verlag (1998)

[Shapiro 91] Shapiro, E.: Inductive Inference of Theories from Facts, in Lassez, J.-L. and Plotkin, G. eds., *Computational Logic*, pp. 199–254, MIT Press, Cambridge, MA (1991)

[Yamamoto 00a] Yamamoto, A.: Using Abduction for Induction Based on Bottom Generalization, in Flach, P. A. and C.Kakas, A. eds., *Abduction and Induction*, Applied Logic Series 18, pp. 267–280, Kluwer Academic Publishers (2000)

[山本 00b] 山本 章博, 有村 博紀, 平田 耕一：帰納論理プログラミングと証明補完, 森下 真一, 宮野 悟（編）, bit 別冊 発見科学とデータマイニング, pp. 34–44, 共立出版 (2000)