Алгоритм SNLP

Алгоритм SNLP для нахождения плана P при заданном множестве целевых предусловий A можно представить на основе обобщенного алгоритма *Refine-Plan*. Схема алгоритма SNLP приведена на рис.1.1 и 1.2. Алгоритм SNLP вызывает следующие функции и процедуры:

Функция *SNLP_Sol(P,G):*

Если множество A пусто, возвратить решение. Если множество A не пусто и длина плана не превысила 1_m ,возвратить значение «продолжить». Если множество А не пусто и длина плана превысила l_m , возвратить значение «нет решений».

Процедура *SNLP_pick-prec:*

Произвольным образом выбрать из множества целевых предусловий A цель $<c,t>$ (где c – предусловие оператора, соответствующего шагу t). Убрать цель $<c,t>$ из множества целевых предусловий A ($A = A - <c,t>$).

Процедура *SNLP_establish:*

Выбрать существующий или новый шаг t' для $<c,t>$, устанавливающий условие c перед шагом t (если шаг t' не существует и не может быть добавлен, вернуться в предыдущую точку возврата). (1) Дополнить множество шагов T шагом t $(T = T \cup \{t\})$. Дополнить множество упорядочивающих ограничений O ограничением $\{t' \Rightarrow t\}$, гарантирующим выполнение шага t' перед шагом t $(O = O \cup \{t' \Rightarrow t\})$. (2) Дополнить множество ограничений инициализаций переменных B множеством B', состоящим из совместных и несовместных инициализаций, содержащихся во множестве *precond(t')* – множестве предусловий выполнения шага t' $(B = B \cup B')$. (3) Для каждого шага t'', выполняемого между t' и t и нарушающего условие c, выполнить два уточнения плана: первое – с помощью ограничения $O = O \cup \{t \Rightarrow t''\}$, и второе – с помощью ограничения $O = O \cup \{t'' \Rightarrow t'\}$ Если шаг t' ранее не встречался в плане, то дополнить множество целевых предусловий A и множество дополнительных ограничений L:

$A = A \cup \{< c', t' >\} \mid \forall c' \in precond(t')\}$;

$L = L \cup \{< c'@t' >\} \mid \forall c' \in precond(t')\}$.

Процедура *SNLP_save:*

Добавить вспомогательные ограничения, гарантирующие сохранность сделанного уточнения плана с использованием стратегии защиты с помощью уточнений (contributor protection strategy), которая обеспечивает систематичность поиска):

$L = L \cup < t', c, t > \cup < t', \neg c, t >$.

Процедура *SNLP_tract:*

Используется способ 2.b, позволяющий разрешать конфликты. (1) Определить конфликты. Шаг t_{conf} считается конфликтующим с IPC-ограничением $< t_1, p, t_2 >$, если $t_{conf} \in T, < t_1, p, t_2 > \in L$, причем шаг t_{conf} может быть выполнен между шагами t_1 и t_2, и выполнение t_{conf} нарушит условие p.

(2) Разрешить конфликты. Для каждого конфликта, состоящего из шага t_{conf} и IPC-ограничения $< t_1, p, t_2 >$, произвести два уточнения плана, первое – с помощью ограничения $O = O \cup \{t \Rightarrow t_1\}$, а второе – с помощью ограничения $O = O \cup \{t_2 \Rightarrow t\}$.

Функция *SNLP_check:*

Если план нецелостный, возвратить значение «нецелостный план», иначе – возвратить значение «целостный план». Нарушением целостности плана является 1) наличие циклов в порядке следования операторов (проверка множества упорядочивающих ограничений O на наличие замкнутых цепочек вида: $\{t_i \Rightarrow t_j \Rightarrow \mathrm{L} \Rightarrow t_i\}$ или 2) наличие для пары переменных одновременно совместной и несовместной инициализации (проверка множества ограничений инициализаций переменных B).

Алгоритм *SNLP*([*P*:[*T*,*O*,*B*,*ST*,*L*],*A*])

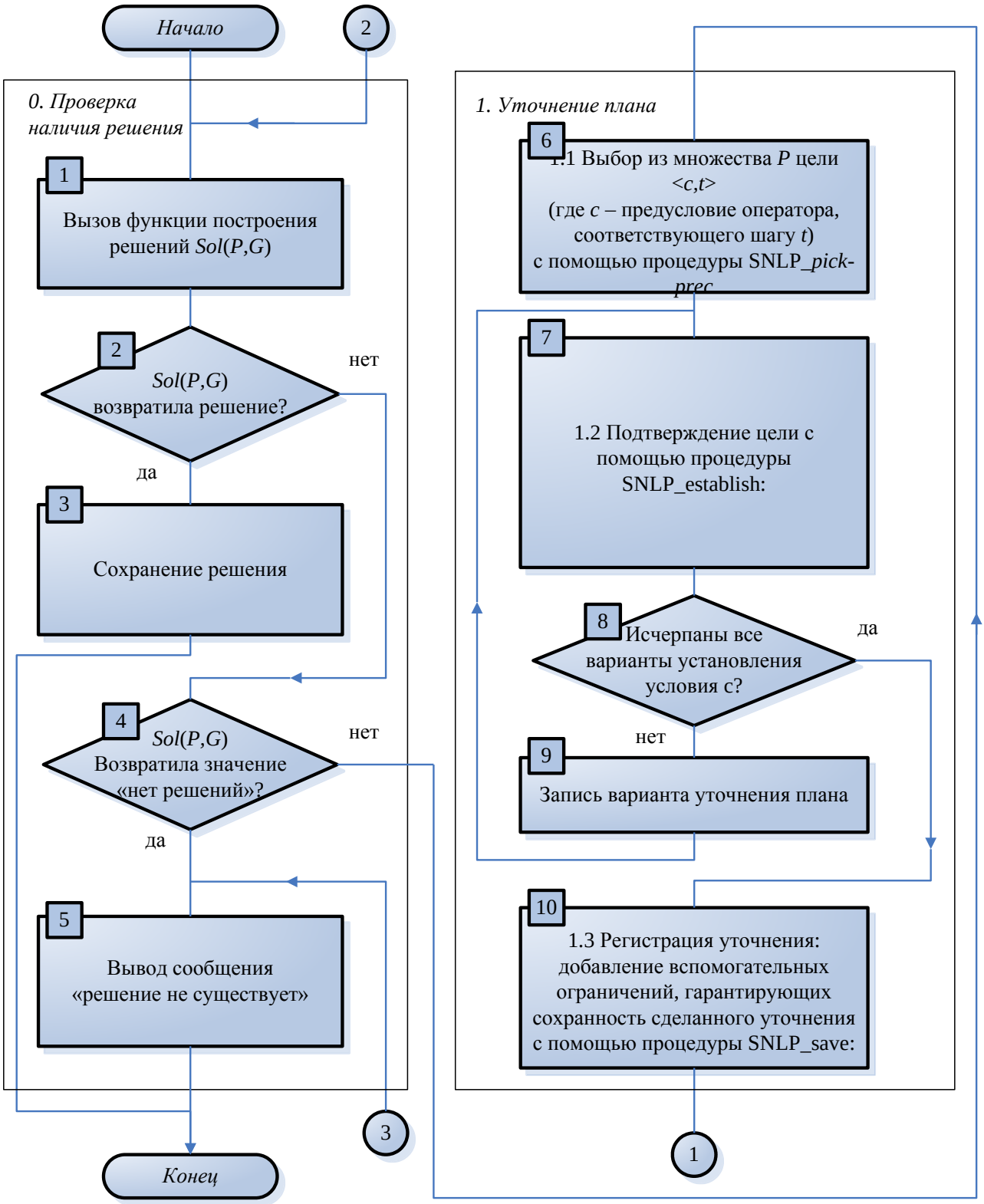

рис.1.1

Алгоритм *SNLP*([*P*:[*T*,*O*,*B*,*ST*,*L*],*A*])

(продолжение)

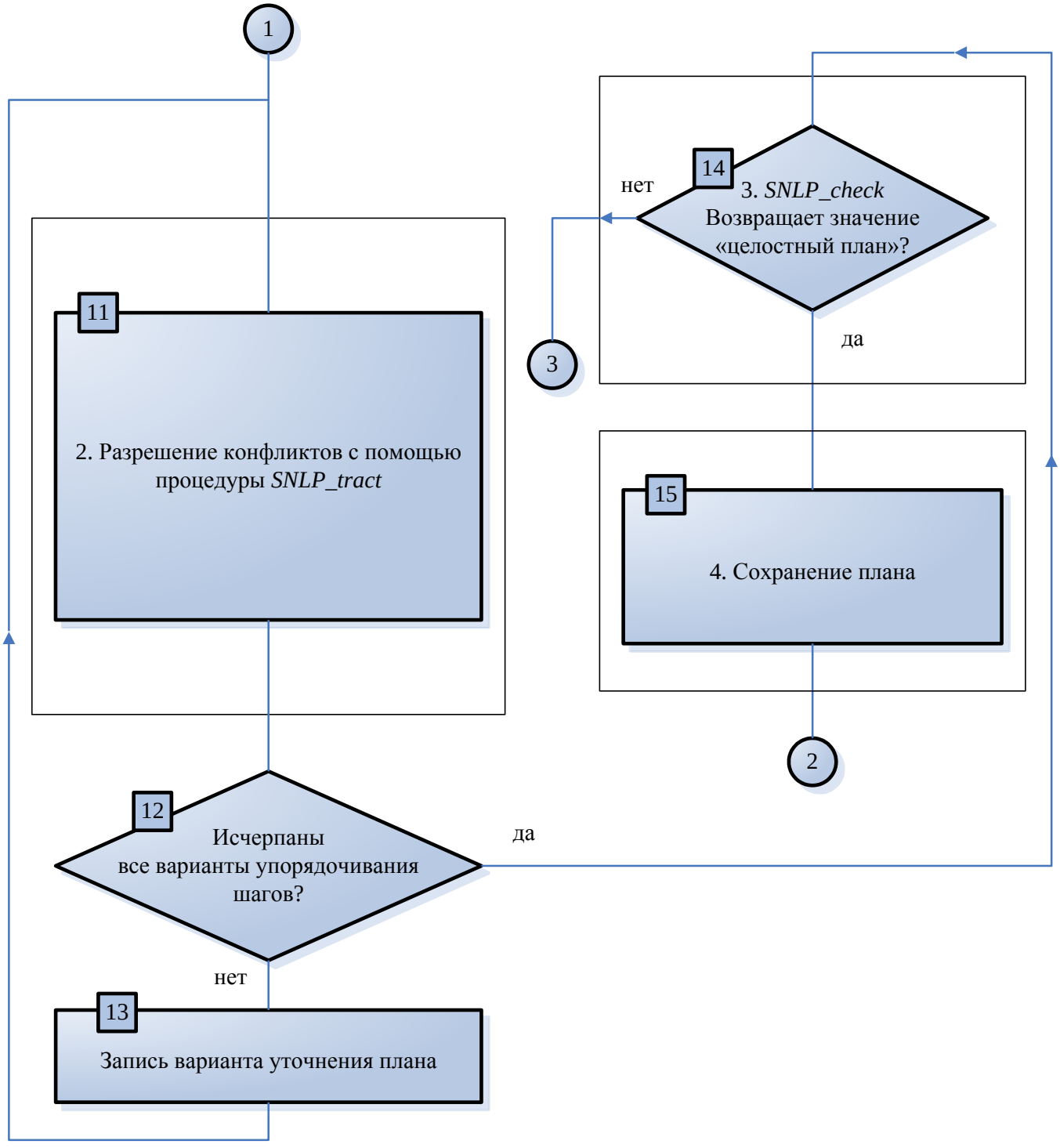

рис.1.2

Алгоритм NONLIN

Алгоритм SNLP является развитием семейства алгоритмов NONLIN. Принципиальным различием между NONLIN и SNLP является использование ими различных стратегий при регистрации уточнений (шаг 1.3). В NONLIN используется стратегия защиты интервала (interval preservation strategy). Следует заметить, что используемая в NONLIN стратегия может приводить к более быстрому нахождению решений для некоторых задач, так как стратегия проверки целостности SNLP может приводить к более частым возвратам. Но для других задач поиск с помощью NONLIN может быть более долгим и избыточным (используемая стратегия не гарантирует систематичности поиска). Для анализа отличий алгоритмов при решении задач планирования используется McNONLIN – разновидность алгоритма NONLIN, описание которого на основе обобщенного алгоритма Refine-Plan полностью идентично описанию алгоритма SNLP за исключением шага 1.3. При регистрации уточнений на шаге 1.3. вместо процедуры *SNLP_save* вызывается процедура *NONLIN_save.*

Процедура *NONLIN_save:*

Добавить вспомогательные ограничения, гарантирующие сохранность сделанного уточнения плана с использованием стратегии защиты с помощью уточнений (contributor protection strategy), которая не обеспечивает систематичность поиска:

$$L = L \cup < t', c, t > .$$

Алгоритм TWEAK

Основное отличие TWEAK от SNLP и NONLIN заключается в том, что TWEAK не использует ограничения для регистрации уточнений плана. Таким образом, TWEAK не задействует ни шаг 1.3., ни шаг 2 обобщенного алгоритма. Кроме того, TWEAK использует МТС-критерий для оценки того, что является ли целевое предусловие $< c,t > \in A$ необходимо истинным (то есть справедливо для всех элементарных линеаризаций плана). При выборе цели (шаг 1.1.) предпочитаются целевые предусловия, для которых не выполняется МТС-критерий. Проверка наличия решения (шаг 0.) завершается успешно в том случае, если МТС-критерий выполняется для всех целевых предусловий из множества *A*. Схема алгоритма TWEAK на основе обобщенного алгоритма Refine-Plan [39] приведена на рис. 3.1 и 3.2. Далее приводятся описания процедур и функций, вызываемых алгоритмом TWEAK и отличных от процедур и функций, приведенных в первом случае.

Функция *TWEAK_Sol(P,G):*

Если каждая цель $< c,t >$ (где c – предусловие оператора, соответствующего шагу t) из множества целевых предусловий A необходимо истинна согласно МТС-критерию, то возвратить решение. Если нет и длина плана не превысила l_m, то возвратить значение «продолжить». Если нет и длина плана превысила l_m, то возвратить значение «нет решений».

Процедура *TWEAK_pick-prec:*

Произвольным образом выбрать из множества целевых предусловий A цель $< c,t >$, для которой не выполняется МТС-критерий. Не удалять цель $< c,t >$ из A.

Из описания следует, что TWEAK не исключает рассматриваемые цели из множества целевых предусловий *A* и поэтому теоретически может рассмотреть одно и то же целевое предусловие более одного раза. Кроме того, хотя TWEAK не уменьшает вычислительную сложность (не задействует шаг 2), за счет использования МТС-критерия алгоритм сохраняет свойство полноты.

Алгоритм *TWEAK*([*P*:[*T*,*O*,*B*,*ST*,*L*],*A*])

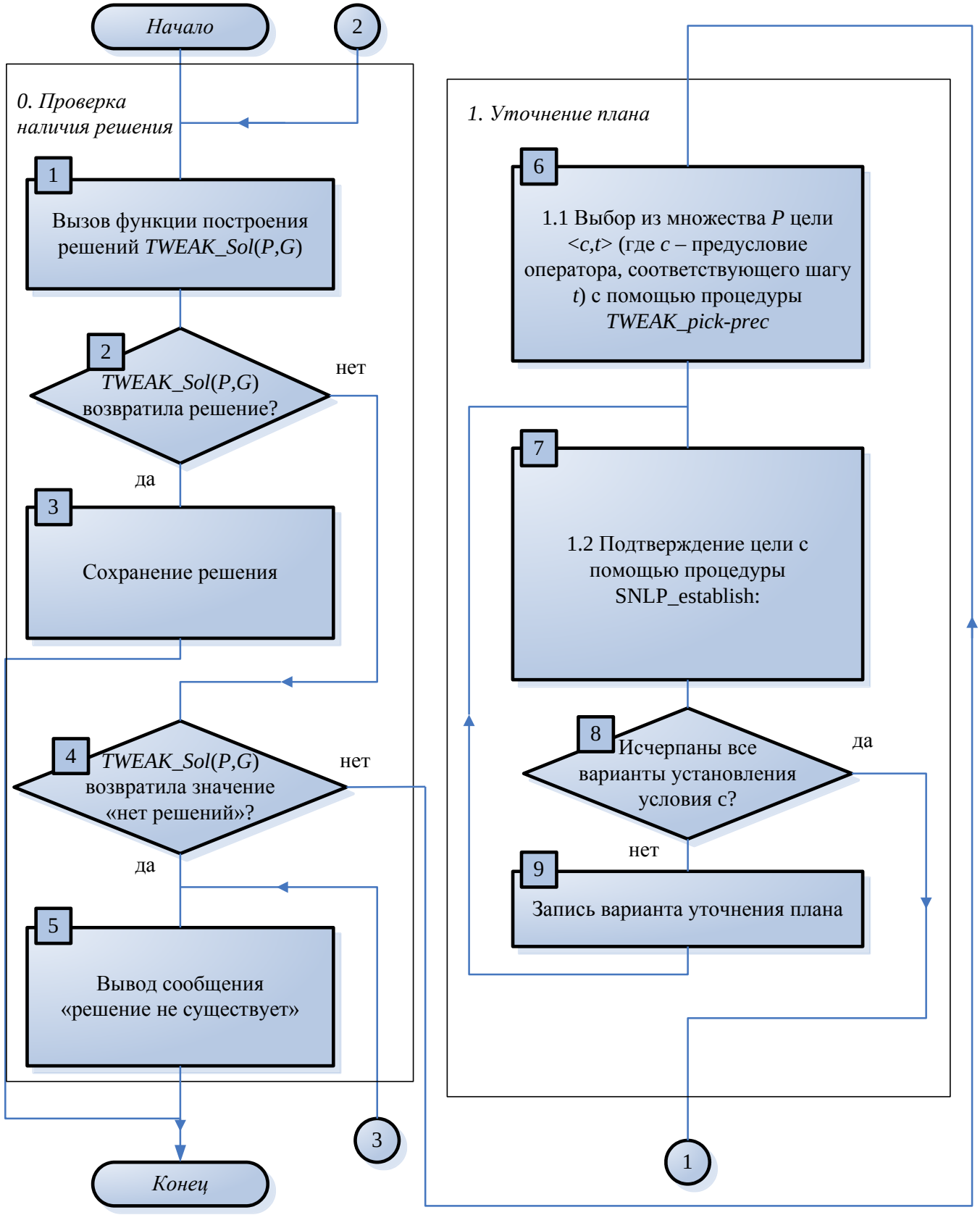

рис.3.1

Алгоритм *TWEAK*([*P:*[*T*,*O*,*B*,*ST*,*L*],*A*])
(продолжение)

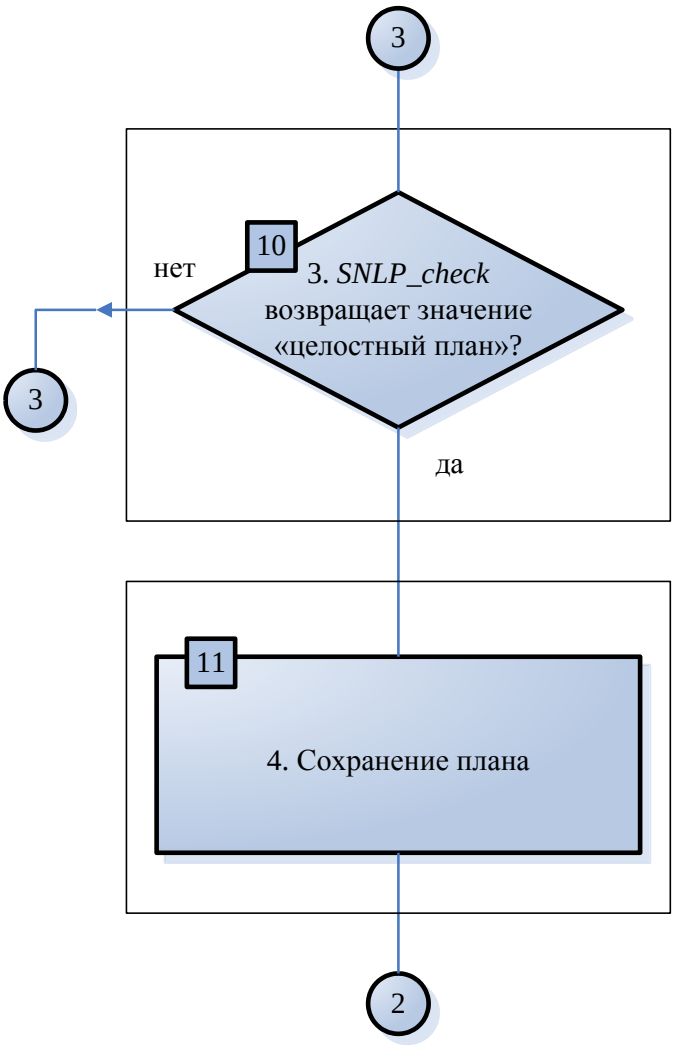

рис.3.2

Алгоритм UA

UA во многом аналогичен алгоритму TWEAK, Он также не регистрирует уточнения плана и использует аналогичные стратегии выбора цели и проверки наличия решения. Принципиальным отличием между UA и TWEAK является только наличие в UA шага уменьшения вычислительной сложности (шаг 2.), на котором в UA используется предварительное упорядочивание (шаг 2.а. обобщенного алгоритма). Таким образом, для описания UA на основе обобщенного алгоритма достаточно взять за основу описание алгоритма TWEAK, приведенное в Приложении 3, и заменить процедуру, вызываемую на шаге 2 для уменьшения вычислительной сложности, на процедуру *UA_tract.*

Процедура *UA_tract:*

Используется способ 2.а. – предварительное упорядочивание. (1) Определение пересечений. Пара шагов плана t_1 и t_2 считаются пересекающимися, если они не упорядочены относительно друг друга (множество O не содержит ни упорядочивающего ограничения $\{t_1 \Rightarrow t_2\}$, ни упорядочивающего ограничения $\{t_2 \Rightarrow t_1\}$ и выполняется одно из следующих трех условий:

(I) t_1содержит предусловие p, а t_2 содержит постусловие p или $\neg p$;

(II) t_2 содержит предусловие p, а t_1 содержит постусловие p или $\neg p$;

(III) t_1 содержит постусловие p, а t_2 содержит постусловие $\neg p$.

На шаге 2.а. (1) требуется найти все шаги t', пересекающиеся с шагом t.

(2) Разрешение пересечений. Для каждого шага t', пересекающегося с шагом t, требуется добавить либо упорядочивающее ограничение $\{t \Rightarrow t'\}$, либо упорядочивающее ограничение $\{t' \Rightarrow t\}$. Для гарантирования полноты поиска необходимо рассмотреть оба варианта.

Из описания стратегии UA на шаге 2. следует, что все частично-упорядоченные планы, генерируемые UA, являются однозначными *(unambiguous),* то есть каждое целевое предусловие $< c,t >$ либо необходимо истинно (то есть справедливо для всех элементарных линеаризаций плана), либо необходимо ложно (то есть не выполняется для всех элементарных линеаризаций плана). В

связи с этим для проверки МТС-критерия в UA достаточно исследовать только одну элементарную линеаризацию плана.