Б. К. КАРАПЕТЯН

ЯЗЫК ВЫСОКОГО УРОВНЯ ДЛЯ ОПИСАНИЯ СТРАТЕГИЙ В ИГРАХ

Введение

Понятие стратегии для позиционных игр определяется как некоторое отображение (вообще говоря не однозначное), множества позиций в ходы. В классической теории игр в основном исследуются условия существования оптимальных стратегий без рассмотрения вопросов их практического использования, в частности, вопросов представления, порождения и хранения стратегий. В то же время именно эти и аналогичные вопросы составляют предмет исследования в программировании игр, где каждая программа, по существу, является конкретной реализацией некоторой стратегии.

Большинство существующих игровых программ используют при выборе хода минимаксную процедуру перебора дерева игры с конкретной оценочной функцией, которая по существу является некоторым квазианалитическим описанием используемой стратегии. При таком подходе возникают трудности модификации стратегий, связанные с большими размерами дерева игры, с невозможностью точного описания структуры позиции посредством оценочной функции.

В последнее время появился ряд работ, в которых исследуется возможность представления и использования стратегий на основе систем закономерностей [1] (продукций) [7, 8] типа <класс позиций⟶рекомендуемые действия>, характерных для шахматиста-мастера и существенно сокращающих порождаемые деревья перебора (в частности для эндшпилей). Так в [7] рассмотрена возможность применения системы AL/1 для шахматных эндшпилей. Основу систем AL/1 составляют «таблицы советов» (advice tables), которые состоят из правил следующего типа:

ЕСЛИ <предусловие> ТО <список рекомендаций>.

Для шахматных эндшпилей <предусловие> выделяет некоторый класс позиций, а <список рекомендаций> описывает возможные ходы в этом классе. Использование таблиц советов для выбора хода выполняется простой процедурой: найти первое правило, предусловие которого истинно для текущей позиции, и выполнить соответствующие ходы. Такая управляющая структура оказалась недостаточной для сложных эндшпилей и в [8] предложена система AL3, в рамках кото-

рой можно описывать планы и цели, позволяющие управлять процессом выбора очередного правила.

Более универсальный подход предложен в [9], где описывается программа поиска комбинаций, приводящих к выигрышу материала. Планы, порождаемые этой программой, задают последовательность действий, которые необходимы для достижения конкретной цели. Реализация каждого действия включает в себя рассмотрение ответов противника, порождение возможных подпланов, определение так называемых опасных ходов, которые необходимо опровергнуть, и построение планов опровержения таких ходов. При построении стратегий программа производит относительно небольшой перебор неперспективных вариантов. Однако ограничение списка возможных целей и использование для задания искомой комбинации материального выигрыша ограничивают область применения программы именно комбинациями, усложняют ее модификацию для построения более общих выигрышных стратегий.

Шахматная программа PARADISE, использующая для управления поиском хода базу знаний, описана в [10]. Построение стратегии игры основано на статическом анализе позиции, результатом которого является дерево планов, используемое в дальнейшем как основа искомой стратегии. Каждый узел дерева планов либо определяет конкретный ход, либо описывает цель, для достижения которой необходимо обращение к базе знаний. База знаний содержит возможные цели и действия по их реализации в обобщенной форме — так называемые «источники знаний», которые конкретизируются при использовании в текущей позиции. Если построенное дерево планов не реализуется программой, то анализ выполненных действий с использованием многоступенчатой оценочной функции выделяет некоторую позицию, в которой заново производится построение планов. В указанных выше работах [7—10] не рассмотрена возможность непосредственного управления процедурой реализации планов: последовательность шагов реализации стратегий либо зафиксирована [7, 8], либо определяется единообразной процедурой [9, 10].

В отличие от них в настоящей работе описываются средства использования при построении стратегий не только декларативных знаний типа закономерностей (продукций), но и процедуральных знаний, описывающих определенные комбинации выполняемых действий. Стратегия задается на алгоритмическом языке высокого уровня (SDL) в виде конкретной процедуры, функциональными операторами которой являются закономерности. Реализация заданной стратегии в некоторой позиции осуществляется многоуровневой компиляцией: интерпретацией SDL—описанной стратегии в закономерности и трансляцией каждой закономерности в шахматные ходы.

1. ПРЕДСТАВЛЕНИЕ И ИСПОЛЬЗОВАНИЕ СТРАТЕГИЙ В ПОЗИЦИОННЫХ ИГРАХ

Позиционная игра двух противников определяется следующими компонентами:

а) множеством позиций $\mathbf{P}$,

б) множеством ходов M, которые задают переходы от одной позиции к другой;

в) множеством начальных позиций $\mathbf{P}_0 \subset \mathbf{P}$;

г) множеством заключительных позиций $\mathbf{P}_f \subset \mathbf{P}$, для каждой из которых заданы выигрыши игроков.

Игру такого типа можно описать деревом $G = (V, E)$, где множество вершин V соответствует множеству позиций игры $\mathbf{P}$, а множество ребер E соответствует ходам. Если вершинам $v_1, v_2 \in V$ соответствуют позиции $p_1, p_2 \in \mathbf{P}$, то ребро $(v_1, v_2) \in E$ тогда и только тогда, когда существует ход $m \in M$, переводящий p_1 в p_2. На концевых вершинах дерева указаны выигрыши каждого игрока. Мы будем рассматривать конечные игры двух противников с полной информацией и нулевой суммой. Для дерева G выполняются следующие условия:

а) в дереве G нет бесконечных ветвей;

б) в каждой вершине дерева указано, какому из игроков принадлежит право хода;

в) выигрыш одного из игроков в заключительной позиции равен проигрышу другого.

Игроков условно мы будем называть игрок 1 и игрок 2.

Каждая ветвь дерева задает некоторую партию, результат которой определяется соответствующей заключительной позицией. Мы будем рассматривать игры, в которых результат партии может быть трех типов: выигрыш, ничья или проигрыш игрока, сделавшего первый ход в партии.

Стратегией игрока i называется функция $S^i : V^i \to E$, где $V^i \subset V$, множество позиций с ходом игрока i. Здесь и далее $i, j \in \{1, 2\}$, $i \neq j$. Ход m игрока i в позиции p определен стратегией S^i, если в вершине $v \subset V^i$, соответствующей p, ребро $s^i(v)$ соответствует ходу m.

Если задана позиция p и стратегии S^1 и S^2 игроков 1 и 2, то однозначно определяется партия $\pi(S^1, S^2, p) = \langle p, p_1, p_2, \ldots, p_k \rangle$, в которой позиции $p_1, p_2, \ldots, p_k$ получаются ходами, определенными стратегиями S^1 и S^2.

Пусть p — позиция с ходом игрока i. Стратегия S^i называется выигрышной, ничейной или проигрышной для игрока i, если для любой стратегии S^j игрока j $(j \neq i)$ партия $\pi(S^i, S^j, p)$ является соответственно выигрышной, ничейной или проигрышной.

Выигрышность стратегии является достаточным обоснованием для выбора хода. Если для некоторого множества позиций $\mathbf{P}_s$ игрок i име-

ет выигрышные стратегии, то в позициях, расположенных выше на дереве игры он будет, по возможности, выбирать ходы, приводящие к позициям из P_s.

Во всех нетривиальных играх деревья игры имеют необозримые размеры, что затрудняет выбор хода. Использование стратегии с известными свойствами уменьшает объем рассматриваемой информации, позволяя оперировать при анализе позиции не отдельными ходами, а поддеревьями и классами позиций.

Ниже проблема представления и использования стратегий рассматривается на примере шахмат.

Одним из способов представления стратегий является ее аналитическое задание, т. е. определение хода по некоторой формуле, учитывающей особенности рассматриваемой позиции. Такой способ задания стратегии используется во всех современных игровых программах, использующих оценочную функцию.

Оценочная функция является либо линейной комбинацией некоторого набора предикатов, включенных с весовыми коэффициентами, либо суперпозицией линейных и логических функций. Использование оценочной функции предполагает ее вычисление для всех позиций, возникающих из данной, после чего выбирается ход, приводящий в позицию с наилучшей оценкой (максимальной для белых, минимальной для черных). Предикаты, используемые в оценочной функции, выделяют различные существенные признаки позиций, такие как ценность фигур и их расположение, слабые и сильные поля и удары на них и т. д. Сложность использования шахматных понятий в оценочной функции заключается в том, что оценочная функция должна быть быстровычисляемой и компактной, а так как при каждом обращении к оценочной функции должны вычисляться все предикаты, то каждый из них должен быть относительно простым. Это вынуждает заменить признаки более простыми, не совпадающими с их формальными аналогами. Кроме этого в шахматной литературе одно и то же понятие используется по-разному в зависимости от стадии игры, от текущих планов и целей. Все эти аспекты использования должны моделироваться в оценочной функции весами предикатов, что трудно осуществимо. Все это приводит к тому, что ход, выделенный оценочной функцией не всегда является хорошим в данной позиции. Для обеспечения точности оценки ее использование дополняется переборной минимаксной процедурой, которая строит дерево игры на определенную глубину, вычисляет оценку заключительной позиции и выполняет подъем оценки по уровням дерева по принципу минимакса. Точность полученной оценки зависит от глубины дерева перебора, которую при ограниченных ресурсах можно увеличить отсечением неперспективных ветвей дерева, поэтому сила игры переборных программ существенно зависит от эвристик, применяемых при таких отсечениях.

Влияние различных предикатов оценочной функции на выбор хода переборной процедурой усредняется, поэтому очень сложно определить необходимость изменения весов признаков, их удаления или добавления.

Вопросы накопления знаний, синтеза новых стратегий, использования результатов предыдущих партий посредством оценочной функции также сводятся к изменению весов предикатов и самих предикатов и остаются неразрешенными (такие решения как «служба лучших ходов», «служба лучших ответов» [1] реализованы вне оценочной функции).

Другой способ представления стратегии, требующий минимальных временных ресурсов при использовании, является задание стратегии деревом (p-стратегии) [2].

Если задана позиция p с ходом игрока i, то стратегию S^i можно описать поддеревом G^i дерева G, которое строится следующим образом:

а) Корнем поддерева G^i является вершина v, соответствующая позиции p;

б) Если некоторая вершина v_1 с ходом игрока i включена в G^i, то в G^i включается также ребро $S^i(v_1) = (v_1, v_2)$ и вершина v_2 с ходом игрока j;

в) Если некоторая вершина v_1 с ходом игрока j включена в G^i, то в G^i включаются также все ребра $(v_1, v_{21}), (v_1, v_{22}), \ldots, (v_1, v_{2l})$ которые исходят из вершины v_1 в дереве игры G и вершины $v_{21}, v_{22}, \ldots, v_{2l}$ с ходом игрока i.

Другими словами из вершин дерева G^i с ходом игрока i исходит ровно одно ребро, соответствующее ходу, определенному стратегией S^i, а из вершины с ходом игрока j исходят ребра, соответствующие всем ходам из этой позиции. Для использования поддерева G^i при выборе хода, достаточно иметь указатель на текущую позицию, который определяет ход игрока i в позициях с его ходом и при каждом ответе противника передвигается на соответствующую вершину. Существенный недостаток такого представления — это размерность соответствующих деревьев. Даже для такого простого эндшпиля, как «Король, ладья против короля», дерево стратегии для большинства позиции имеет порядка 10^{10} вершин. Более сложные стратегии приводят к экспоненциальному росту размеров деревьев, и проблемы порождения, хранения и модификации стратегий при таком представлении становятся практически неразрешимыми.

Уменьшения требуемых ресурсов памяти можно достигнуть, используя для описания стратегий понятия, более информативные, чем ход. Такие понятия должны задавать преобразования классов позиций, обладающих некоторыми общими свойствами, в другие классы. Формализация таких преобразований определена в [2] как $<P, F>$-стратегии и их слияния, где P — исходная позиция, и F — описание класса позиций, достижимых $<P, F>$-стратегией. Рассмотрение шахматной литературы показывает широкое использование преобразований этого типа как при обучении игре, так и при анализе и разборе партии. Стратегии при этом описываются в таких терминах, как «оттеснение короля», «фланговая атака», «игра на противоположном фланге», «исполь-

зование слабых полей» и т. д. При этом в большинстве случаев описание стратегии игры в конкретных классах позиций задается алгоритмически. Примером может служить приводимое ниже описание стратегии игры в эндшпиле «Король, ладья против короля»:

А) Отсечь короля противника ладьей;

Б) Если короли находятся в оппозиции и ладья может дать шах с безопасного поля, то оттеснить короля к краю доски шахом; если мата нет, то перейти к В), иначе цель достигнута;

В) Если оппозиция возможна после хода противника, то сделать выжидательный ход ладьей и перейти к Б);

Г) Если король противника отступил от линии отсечения, то оттеснить короля ладьей, перейти к Б);

Д) Если короли находятся дальше, чем на 2 хода друг от друга, то подойти своим королем к королю противника, не переходя линию отсечения; перейти к Б);

Е) Во всех случаях, если король противника атакует ладью, отступить ладьей по линии отсечения на противоположный край доски.

Аналогичные описания можно получить для стратегий игры не только во многих других форсированных окончаниях, но и для представления стандартных комбинационных приемов игры (завлечение фигуры под связку, атака связанной фигуры, «мельница», «спертый мат» и т. д.).

Рассмотрение таких примеров показывает, что в шахматной теории и практике стратегии описываются в терминах более элементарных действий, способ реализации которых ходами известен, либо описан ранее. Само описание стратегии является алгоритмом, посредством которого можно получить последовательность элементарных действий, необходимых для реализации стратегии. Алгоритмическое представление стратегий по существу является процедурой на языке высокого уровня, задающей последовательность преобразований позиций, в терминах элементарных действий. Использование такой процедуры возможно с помощью транслятора, который обеспечивает преобразование <позиция> → <ход>. В данной работе описывается язык описания стратегий SDL (Strategy Description Language), предназначенный для процедурного представления стратегий и SDL — интерпретатор, обеспечивающий реализацию SDL — стратегии для заданной позиции.

2. ОПИСАНИЕ ЯЗЫКА

Язык *SDL* позволяет задавать T-слияние стратегий [2], используя фиксированный набор элементарных действий (здесь T — это множество позиций, в которых применима данная стратегия).

Входными данными для *SDL*-стратегий (т. е. стратегии, заданной на языке *SDL*) является исходная позиция с ходом белых. *SDL*-стратегия по существу является процедурой, которая задает порядок применения элементарных действий в исходной позиции p, необходимый для построения дерева p-стратегии. Элементарные действия составляют

функциональные операторы языка *SDL*, т. е. операторы, задающие преобразование позиций. Порядок применения элементарных действий задается управляющими операторами языка.

2.1. Представление позиции и классов позиций

Позиция задается как набор фигур, белых и черных, каждая из которых обозначается однобуквенным символом: *K*—король, *Q*—ферзь, *R*—ладья, *B*—слон, *N*—конь и *P*—пешка. Описание позиций задается посредством досок. Каждая доска — это матрица размерности 8×8 нулей и единиц. Для досок определены операции объединения и пересечения, которым соответствует покомпонетная дизъюнкция и конъюнкция соответственно, т. е. если $D_1 = \{d^1_{i,j}\}$, $D_2 = \{d^2_{i,j}\}$ и $D_3 = \{d^3_{i,j}\}$ — доски, то для объединения $D_1 \cup D_2 = D_3$ имеем $d^3_{ij} = d^1_{ij} \vee d^2_{ij}$, а для пересечения $D_1 \cap D_2 = D_3$ — $d^3_{ij} = d^1_{ij} \& d^2_{ij}$. Выражение, составленное из объединения и пересечения досок, налывается первичным выражением. Два первичных выражения, соединенных символом отношения, составляют элементарный дескриптор. В SDL используются отношения $=$, $\neg =$, $\subset$. Элементарный дескриптор — это предикат, равный единице, если отношение выполняется, и нулю в противном случае. Отношение равенства (неравенстева) выполняется, если левая и правая части покомпонентно совпадают (не совпадают). Отношение включения выполняется, если выполняется компонентная импликация. Выражение, составленное конъюнкцией и дизъюнкцией элементарных дескрипторов, задает дескриптор класса позиций, т. е. предикат, равный единице для позиций описываемого класса и нулю для остальных позиций.

Элементарные дескрипторы используются для описания шахматных понятий, таких как открытая вертикаль, проходная пешка, оппозиция королей и т. д. Исследование дескрипторов, описанных в [3], позволило выделить набор досок, посредством которых можно описать соответствующие классы позиций. В язык *SDL* включены доски трех типов:

1) геометрические доски, описывающие такие понятия, как вертикаль, горизонталь, диагональ, квадрат проходной пешки, центр доски, королевский фланг, и т. д.

2) фигурные доски, описывающие понятия, связанные с расположением одной или группы фигур, например, доска положения фигуры (единице в клетке, занятой данной фигурой, и нули в остальных клетках); доска оппозиции королей, доска критических полей проходной пешки и т. д.

3) доски возможностей, описывающие понятия, связанные с ходами фигур, т. е. в некотором смысле динамические. Сюда относятся доски задающие ходы фигур, описывающие связку, вскрытый шах, «вилку» и т. д

Каждая доска в *SDL* задается уникальным именем и списком параметров, заключенных в круглые скобки. В качестве параметров досок выступают обозначения фигур. Например: выражение *SQ* (*W.R*) означает доску расположения белой ладьи; *R* (*W.R*) — это обозначение горизонтали, на которой стоит белая ладья (т. е. единицы по описываемой горизонтали и нули на остальных клетках).

2.2. Описание элементарных действий

Элементарное действие задается следующей синтаксической конструкцией:

ACTION (< имя действия >, < исходный класс >, < конечный класс >).

Здесь ACTION—ключевое слово, выделяющее данное действие;

<имя действия> — идентификатор произвольной длины, обозначающий одно из действий, зафиксированых в языке;

<исходный класс> — дескриптор, задающий условие применимости данного действия, т. е. множество позиций, в которых действие выполнимо;

<конечный класс> — дескриптор, описывающий результат применения данного действия.

Дескрипторы, задающие исходные и конечные классы позиций, вычисляются для той позиции, в которой применяется данная закономерность. Если для некоторой доски, входящей в дескриптор конечного класса, необходимо указать, что она вычисляется после применения данной закономерности, то перед именем этой доски необходимо поставить букву *N* и точку. Например, исходный класс позиций, в которых короли находятся в оппозиции, можно задать дескриптором

$$SQ(W.\ K) \cup SQ(B.\ K) = OPP(W.\ K,\ B.\ K),$$

где $SQ(W.\ K)$, $SQ(B.\ K)$ — доски паложения белого и черного короля, а $OPP(W.\ K,\ B.\ K)$ — доска оппозиции. Тогда конечный класс, после шаха ладьей, можно задать дескриптором

$$SQ(W.\ K) \cup SQ(B.\ K) = OPP(W.\ K,\ B.\ K) \,\&\, N.\ L(W\ R) = L(B.\ K),$$

где L (⟨ фигура ⟩) — доска линии указанной фигуры.

2.3. Управляющие операторы

Управляющие операторы *SDL* определяют порядок выполнения закономерностей в зависимости от типа достигнутой позиции. В *SDL* таких операторов два: оператор цикла типа *DO-WHILE* и оператор условного выбора *IF*. Как известно [4, 5] операторы цикла и условного выбора достаточны для записи любой процедуры. Для языка *SDL* можно показать, что если некоторая стратегия задана как правильная блок-схема [4], функциональные операторы которой выражаются элемен-

тарными действиями, то ту же стратегию можно записать с помощью операторов цикла и условного выбора, используя те же, что и в блок-схеме, предикаты и соответствующие закономерности. Выбор всего двух управляющих операторов продиктован стремлением к упрощению транслятора *SDL*. Дальнейшее расширение языка предполагает добавление других удобных структур управления.

Оператор *DO-WHILE* состоит из заголовка цикла, тела цикла и конца цикла. Заголовок цикла имеет вид *DO-WHILE* (<дескриптор>), где <дескриптор> задает класс позиций, для которых выполняется тело данного цикла.

Тело цикла составляют произвольные операторы *SDL*. Конец цикла задается в виде *END-DO*.

Если при выполнении стратегии встретился оператор цикла, то дальнейшее выполнение происходит следующим образом:

1. Для полученной на данном этапе позиции вычисляется дескриптор заголовка цикла. Если он принимает значение *О*, то выполняемым оператором выбирается оператор, следующий за концом данного цикла. В противном случае выполнение стратегии продолжается с первого оператора цикла.

2. Если при выполнении тела цикла достигнут конец цикла, то выполняются действия, описанные в 1 для полученной позиции.

Иначе говоря, оператор *DO-WHILE* является оператором цикла с предусловием, т. е. с условием, проверяемым перед первым и каждым следующим выполнением тела цикла.

Синтаксис условного оператора задается в следующей форме:

```
IF ⟨ дескриптор ⟩ THEN
   ⟨ THEN — операторы ⟩
{ELSE — IF ⟨ дескриптор ⟩ THEN
   ⟨ THEN — операторы ⟩ }
[ELSE
   ⟨ ELSE — операторы ⟩ ]
END — IF
```

Здесь квадратные скобки выделяют необязательную часть оператора, а фигурные скобки — часть, которая может отсутствовать или быть повторена несколько раз; дескрипторы описывают классы позиций, в которых должны выполняться соответствующие операторы;

<*THEN*-операторы> и <*ELSE*-операторы> — последовательности любых операторов, не содержащие условного оператора. Необходимость вложенных условных операторов исключена, так как вложенность определена синтаксисом условного оператора.

Условный оператор выполняется следующим образом:

1. Для текущей позиции вычисляется дескриптор, стоящий после ключевого слова *IF*. Если он равен единице, то выполняются *THEN*-операторы, после чего рассматривается оператор, следующий за клю-

чевым словом *END-IF*. Если дескриптор условия равен нулю, то *THEN*-операторы пропускаются и рассматривается следующая лексема.

2. Если очередная лексема *ELSE-IF*, то оператор выполняется так же, как описано в 1.

3. Если очередная лексема *ELSE*, то выполняются *ELSE*-операторы, после чего выбирается оператор, следующий за ключевым словом *END-IF*.

2.4. Операторы STRAT и END—STRAT

Оператор *STRAT* задает имя данной стратегии и ее спецификацию, т. е. описание входов и результатов. Синтаксис оператора *STRAT* следующий:

⟨ имя ⟩ : STRAT (IN — CLASS:
FIGURES — W: ⟨ список фигур ⟩ ;
B: ⟨ список фигур ⟩ ;
DESCR — ⟨ описание класса ⟩ ;
OUT — CLASS: ⟨ описание класса ⟩).

Здесь: <имя> — произвольный идентификатор;

<список фигур> — перечисление фигур соответствующего цвета;

<описание класса> — дескриптор, задающий исходный и конечный классы позиций.

Дескрипторы, задающие описания классов, подчиняются правилам, указанным при описании действий (2.2).

Выполнение оператора *STRAT* заключается в проверке выполнимости входных спецификаций для заданной позиции. Если эта проверка успешна, т. е. совпадает фигурный состав и выполняется дескриптор, задающий исходный класс, то выбирается первый оператор, следующий за оператором *STRAT*, в противном случае стратегия не выполняется.

Оператор конца стратегии имеет следующую форму:

END-STRAT <имя> и служит для обозначения конца текста процедуры.

3. КОМПИЛЯЦИЯ SDL-СТРАТЕГИЙ

SDL-стратегия, так же как и процедуры любого языка высокого уровня, для использования на ЭВМ требует перевода на более низкий уровень представления, а именно на уровень, доступный данной вычислительной системе. Существуют две возможности компиляции программ с языка *L*1 на язык *L*2: трансляция и интерпретация. Трансляция программы, заданной на *L*1 заключается в последовательной замене ее операторов эквивалентными последовательностями команд *L*2 и полу-

чении в результате новой процедуры, заданной на языке *L2*, которая и будет исполняться на ЭВМ.

При интерпретации каждый оператор процедуры на *L1* анализируется и непосредственно реализуется эквивалентной последовательностью операторов языка *L2*. Каждый способ компиляции имеет свои преимущества и недостатки. Трансляция требует больших временных затрат для построения результатной процедуры, однако, выполнение полученной процедуры обычно требует меньше времени, чем при интерпретации исходной процедуры. При трансляции неизбежны затраты памяти для хранения полученной программы, при интерпретации память выделяется только для компиляции отдельного оператора интерпретируемой процедуры. Многоуровневая организация процедуры компиляции [6] позволяет более гибко использовать различные режимы компиляции поэтапным переходом с уровня на уровень.

3.1. Уровни представления SDL-стратегии в компиляторе

Многоуровневая организация компиляции характеризуется несколькими уровнями представления стратегий и процедурами компиляции с уровня на уровень. На каждом уровне исходная стратегия описывается на некотором языке, на верхнем уровне — исходный язык представления стратегии, на самом нижнем уровне — язык, доступный вычислительной системе. Для компиляции *SDL*-стратегии наивысший уровень системы — это язык *SDL*. При исполнении *SDL*-стратегии для заданной позиции последовательно выбирается очередная закономерность, которая должна быть реализована в ходах. Исходя из этого можно выбрать второй уровень описания стратегии — уровень закономерности и следующий уровень — описание стратегии в ходах. Наконец низший уровень системы, доступный вычислительной системе, задается алгоритмическим языком *PL/1*.

На уровне закономерностей *SDL*-стратегию можно представить как дерево исполнения, так называемое *E*-дерево [4], ребра которого соответствуют закономерностям и пути которого изображают все возможные последовательности реализации закономерностей без возвратов назад. При выборе хода на основе *SDL*-стратегии для заданной позиции реализуется конкретная ветвь *E*-дерева, так что рассмотрение оставшихся ветвей дерева становится излишним. Исходя из этого, для компиляции стратегии с уровня *SDL* на уровень закономерностей выбран режим интерпретации.

Реализация закономерности для данной позиции в ходах приводит к *p*-стратегии (см. 1), для построения которой необходимо рассмотрение всех ответов черных в каждой возникающей позиции. По существу для исполнения закономерности необходимо сначала получить полное ее описание в ходах, т. е. необходим трансляторный режим компиляции.

Реализация ходов *p*-стратегии на языке *PL/1*, так же как и реали-

зация *E*-дерева закономерности, естественным образом осуществляется интерпретатором. Описание уровней представления и соответствующих компиляторов приведено на рис. 1.

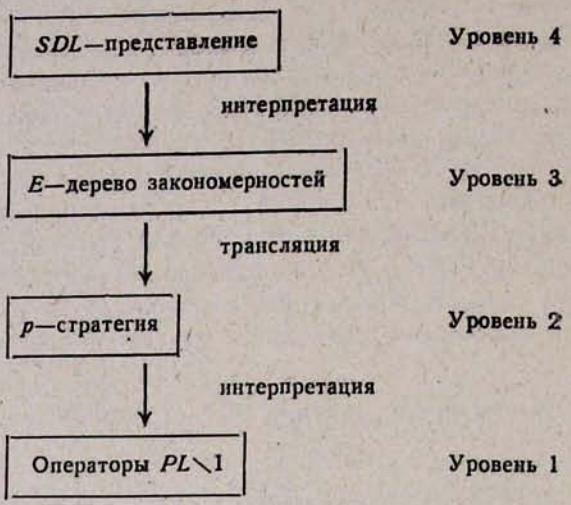

Рис. 1. Многоуровневая организация процедуры трансляции SDL-стратегии на машинный уровень.

3.2. Функционирование SDL-компилятора

Алгоритм работы *SDL*-компилятора приведен на рис. 2 и 3. Для описания алгоритмов мы используем *PL*-подобный псевдоязык [5], в котором управляющие конструкции имеют обычный смысл, а выражения, заключенные в угловые скобки, соответствуют некоторым компонентам или модулям. На рис. 2 выделены основные компоненты компилятора, а на рис. 3 каждая компонента детализована до уровня модуля.

Первая компонента компилятора, инициализация, обеспечивает ввод исходной позиции и текста *SDL*-стратегии, проверяет применимость заданной стратегии в этой позиции и если позиция принадлежит описанию «IN-CLASS» исполняемой стратегии, то подготавливаются необходимые переменные и массивы для последующих модулей.

Если в исходной позиции рассматриваемая стратегия не применима, то компилятор печатает соответствующее сообщение и завершает свою работу. В противном случае работает цикл *DO-WHILE*, при каждой итеррации которого выполняется очередное действие. Компонента «Выделение очередного оператора ACTION», работает в режиме интерпретации и обеспечивает переход с уровня 4 на уровень 3 (рис. 1).

Реализация выделенного действия, (т. е. выполнение оператора ACTION), происходит в модуле «Построение соответствующей *p*-стратегии». Этот модуль работает в режиме трансляции и строит дерево *p*-стратегии, описывающее в ходах выделенное действие (т. е. на уровне 2, рис. 1).

Полученное дерево используется в следующем цикле *DO-WHILE* для реализации заданной стратегии средствами языка *PL/1*. Модуль «Выполнить очередной ход *p*-стратегии» интерпретирует выполнение очередного хода в текущей позиции. Модуль «Получить ответ противника» вводит ответ противника, выполняет его и находит вершину *p*-стратегии, соответствующую ответу на введенный ход. Работа цикла *DO-WHILE* заканчивается при достижении концевой вершины *p*-стратегии, т. е. после реализации очередного действия.

После реализации каждого действия повторяется выполнение внешнего цикла, до тех пор пока не будет достигнут оператор END — STRAT.

```
        SDL — COMPILER: PROCEDURE OPTIONS (MAIN);
    ⟨ Инициализация компилятора ⟩;
    IF ⟨ инициализация не успешна ⟩ THEN
      ⟨ Выдача сообщения о неприменимости ⟩;
    ELSE DO;
DO WHILE (⟨ не достигнут конец стратегии ⟩);
      ⟨ Выделение очередного оператора ACTION ⟩;
      ⟨ Построение соответствующей p — стратегии ⟩;
DO WHILE (⟨ p — стратегия не исполнена ⟩);
      ⟨ Выполнить очередной ход p — стратегии ⟩;
      ⟨ Получить ответ противника ⟩;
            END; /*DO*/
        END; /*DO*/
    END; /*IF*/
END SDL — COMPILER;
```

Рис. 2. Описание SDL — компилятора. Основные компоненты.

```
SDL — COMPILER: PROCEDURE OPTIONS (MAIN);
    /*Инициализация компилятора*/
    ⟨ Ввод позиции POSITION ⟩;
    ⟨ Ввод текста стратегии ⟩;
⟨ Выделение класса применимости ⟩;
    IF ⟨ стратегия не применима ⟩ THEN
      ⟨ выдать сообщение о неприменимости ⟩;
    ELSE DO;
      ⟨ Подготовка таблиц компилятора ⟩;
      ⟨ Выделить очередную лексему стратегии TEMP ⟩
      /*Инициализация завершена успешно*/;
            DO WHILE (TEMP ¬= 'END — STRAT');
          /*Выделение очередного оператора ACTION*/
DO WHILE (TEMP ¬= 'ACTION' & TEMP ¬ = 'END — STRAT');
    IF TEMP = 'DO' ∨ TEMP = 'END — DO' THEN
      ⟨ Обработка оператора цикла ⟩;
```

```
ELSE IF TEMP = 'IF' ∨ TEMP = 'ELSE — IF' ∨ TEMP = 'ELSE' THEN
      ⟨ Обработка условного оператора ⟩;
    ELSE
      ⟨ Выдача сообщения по ошибке ⟩;
END; /*DO*/
    IF TEMP = 'ACTION' THEN DO;
      ⟨ Выделить имя элементарного действия ⟩ ;
      ⟨ Вызов модуля транслятора для построения p — стратегии ⟩;
NODE = ⟨ Корень полученного дерева ⟩;
DO WHILE (⟨ NODE — не концевая вершина p — стратегии ⟩);
      ⟨ Выполнить ход NODE в текущей позиции ⟩;
CORRECT = 'Ø'B;
DO WHILE (¬CORRECT);
      ⟨ Получить ответ черных ⟩;
    IF ⟨ ход правильный ⟩ THEN DO;
      ⟨ Выполнить ход в текущей позиции ⟩;
      ⟨ Найти ход в дереве ⟩;
NODE = ⟨ текущая вершина p — стратегии ⟩;
CORRECT = '1'B;
END; /*THEN*/
    ELSE
      ⟨ Выдача сообщения об ошибке ⟩;
    END; /*DO*/
END; /*DO*/
      ⟨ Выделить очередную лексему стратегииTEMP ⟩
END; /*THEN*/
    ELSE /*либо достигнут конец стратегии, либо ошибка в
                                                стратегии*/;
TEMP = 'END — STRAT';
          END; /*DO*/
    END /*ELSE*/
END SDL — COMPILER;
```

Рис. 3. Алгоритм *SDL* — компилятора. Основные модули.

3.3. Пример SDL-стратегии

Приведем описание *SDL*-стратегии для эндшпиля «Король, ладья против короля», реализующее стратегию, рассмотренную в 1.

SDL-стратегия, названная K — R — VS — K, описывает стратегию белых в указанном эндшпиле, позволяющую оттеснить короля черных к крайней вертикали и объявить ему мат.

```
K — R — VS — K : STRAT
(IN — CLASS :
FIGURES — W : K, R;
          B : K;
```

```
DESCR — ANY;
OUT — CLASS : MV (B. K.) = ∅);
ACTION (LINE, ANY, N. L (W. R) ⊂ DISP (W. K, B. K) &
                N. L (W. R) ⌝ = N. L (W. K))
DO WHILE (MV (B. K) ⌝ = ∅);
IF (OPP (W. K, B. K) & SQ (R) ∩ MV (B. K) = ∅) V L(R) ∩
                MV (B. K) = ∅ THEN
ACTION (NEWLINE, ANY, (N. L (R) = L (B. K) V N. L (R) ⊂
    DISP (W. K, B. K)) & N. SQ (R) ∩ MV B. K) = ∅
ELSE — IF (POPP (W. K, B. K)) THEN
ACTION (DUMMY, ANY, N. L (R) = L (R) & SQ (R) ∩
                AL (R, B. K) ⌝ = ∅)
ELSE IF (SQ (R) ∩ MV (B. K) ⌝ = ∅) THEN
ACTION (ALONG, ANY, N. L (R) = L (R) & SQ (R) ∩
                MV (B. K) = ∅)
ELSE
ACTION (TOKING, ANY, N. DISP (W. K, B. K) ⊂
            DISP (W. K, B. K) & N. L (R) = L (R))
        END — IF;
    END — DO;
END — STRAT.
```

В приведенной стратегии используются следующие доски:

L (FIG) — описание вертикали фигуры FIG;

DISP (FIG1, FIG2) — описание прямоугольника с вершинами в клетках, занятых фигурами FIG1 и FIG2;

MV (FIG) — описание ходов фигуры FIG;

OPP (FIG1, FIG2) — описание вертикальной оппозиции фигур FIG1, FIG2;

ANY — обозначение произвольной доски;

SQ (FIG) — описание положения фигуры;

POPP (FIG1, FIG2) — описание позиций, в которых через ход возможна оппозиция фигур FIG1, FIG2 (предоппозиция [3]);

AL (FIG1, FIG2) — описание позиций, в которых от фигуры FIG1 к фигуре FIG2 можно дойти, сделав более чем 3 хода королем.

Элементарные действия позволяют реализовать следующие преобразования:

LINE — отсечение короля противника по вертикали;

NEWLINE — оттеснение короля противника к краю доски;

DUMMY — промежуточный ход ладьей в случае предоппозиции;

ALONG — отступление ладьи по линии отсечения;

TOKING — оттеснение короля противника своим королем.

Рассмотрим работу *SDL*-компилятора при применении приведенной стратегии для следующей позиции: белые — ладья *g*6, король *e*6, чер-

ные — король c5. Для простоты предложены ответы черных, приводящие к короткой партии.

После инициализации компилятор выделяет действие *LINE* и строит дерево *p*-стратегии глубины 3, с первым ходом Крe5 и для каждого ответа черных с соответствующим ходом белых: 1... Крb5 2. Лc6; 1... Крb4 2. Лc6; 1... Крc4 2. Лb6. Далее выполняется цикл реализации (*PL*-интерпретации) построенной *p*-стратегии в исходной позиции. При этом выполняется и печатается ход Кр e5 и компилятор принимает ответ противника. Предположим черные выбрали ход Кр c4, тогда следующим выполняемым компилятором ходом будет ход Л d6. После того как будет получен ход черных (предположим это ход К b4), *p*-стратегия отсечения короля реализована и компилятор включает модуль «Выделение очередного оператора ACTON», который сканирует текст исходной стратегии. Очередным оператором является оператор *DO-WHILE*. Проверяется дескриптор $MV(B.K) \neg = \varnothing$, и так как он выполняется, то рассматривается тело оператора цикла. В данном случае это условный оператор, и так как $L(R) \cap MW(B.K) = \varnothing$, то в полученной позиции выполнен дескриптор первого *IF*-условия и компилятор выделяет для выполнения действие *NEWLINE*. На этом работа модуля «Выделение очередного оператора ACTION» завершается и компилятор строит дерево *p*-стратегии для реализации выделенного действия. В данном случае оно будет состоять из единственного хода Лc6, который и будет выполнен в текущей позиции. Предположим черные ответили ходом Кр b3. Так как очередное действие реализовано и не достигнут конец стратегии, то компилятор продолжает обработку стратегии. Оператор *END-IF* пропускается, следующий оператор, *END-DO*, требует проверку дескриптора заголовка цикла и так как для полученной позиции он выполняется, то следующим рассматривается условный оператор (первый оператор тела цикла). Ни один из дескрипторов *IF*-оператора не выполняется, что влечет выполнение его *ELSE*-оператора, т. е. действия TOKING, которое реализуется компилятором как ход Кр d4. Если черные ответят Кр b4, то компилятор выдаст ход Л b6, согласно действию при оппозиции NEWLINE (последовательность шагов выделения этого хода, аналогична описанному выше). Таким же образом на ответ черных Кр a5, белые ответят ходом Л b1 (действие ALONG) и если Кр a4, то Кр c4 (действие TOKING). Наконец после ходов Кр b3, Лb8 (действие DUMMY), К a4, Ла 8 X, дескриптор цикла DO — → WHILE $(MV(B.K) \neg = \varnothing)$ не выполняется и следующий за концом цикла оператором стратегии выбирается оператор END — STRAT К — R — VS — К. Обработка этого оператора приводит к завершению работы компилятора, с выдачей записи сыгранной партии: 1. Кр e5 Кр c4 2. Л d6 Кр b4 3. Л c6 Кр b3 4. Кр d4 Кр b4 5. Л b6 Кр a5 6. Л b1 Кр a4 7. Кр c4 Кр a3 8. Л b8 Кр a4 9. Л a8 X.

ԽԱՂԵՐԻ ՍՏՐԱՏԵԳԻԱՆԵՐԻ ՆԿԱՐԱԳՐՄԱՆ ԲԱՐՁՐ ՄԱԿԱՐԴԱԿԻ ԼԵԶՈՒ

Առաջարկված է բարձր մակարդակի լեզու (SDL), որը հնարավորություն է տալիս տարրական օրինաչափություններ միջոցով ներկայացնել «դիրքերի դասերի նկարագրություն—հնարավոր գործողություններ» տիպի օրինաչափությունները։ Դիտարկվում է երկքայլանի ստրատեգիաների սինթեզման պրոցեդուրա, որը SDL-լեզվով տրված օրինաչափությունների հիման վրա կառուցում է համապատասխան ստրատեգիաներ։ Առաջին քայլից հետո ստացվում է կառուցվող ստրատեգիային համապատասխանող տարրական օրինաչափությունների հաջորդականություն, երկրորդ քայլի ընթացքում ամեն մի ստացված տարրական օրինաչափության համար կառուցվում է համապատասխան խաղային ենթածառ։

ЛИТЕРАТУРА

1. *Адельсон-Вельский Г. М., Арлазаров В. Л., Донской М. В.* Программирование игр, М., Наука, 1978.
2. *Погосян Э. М.* Адаптация комбинаторных алгоритмов, Ереван, Изд-во АН АрмССР, 1983.
3. *Погосян Э. М., Амбарцумян М. А., Аджабян Н. А., Арутюнян Ю. Г., Джндоян Л. О., Карапетян Б. К.* Пакет прикладных программ для исследования методов адаптации комбинаторных алгоритмов управления (АКА 1,2. Отчеты ВЦ АН АрмССР за 1976—1980 гг.
4. *Лингер Р., Миллс Х., Уитт Б.* Теория и практика структурного программирования, М., Мир, 1982.
5. *Ван Тассел Д.* Стиль, разработка, эффективность, отладка и испытание программ, М., Мир, 1981.
6. *Таненбаум Э.* Многоуровневая организация ЭВМ, М., Мир, 1979.
7. *Bratko I., Michie D.* A representation for pattern-knowledge in chess end games, Advances in Computer Chess, 2, 31—56, Edinburgh, 1980.
8. *Bratko I.*, Knowlegde-based problem solving in AL3 Machine Intelligence, 10. 73—101, 1981.
9. *Pitrat J.* A chess combination program, which uses plans. Artificial Intelligence, 8, 275—321, 1977.
10. *Wilkins D.* Using knowledge to control tree searching. Artificial Intelligence, 18, 1—51, 1982.