Referat z przedmiotu „Teoria Kompilacji”

Przegląd metod error recovery (dla parsingu top-down, przykłady)

Skąd biorą się błędy ?
Proces obsługi błędów zajmuje się defektami powstającymi z powodu błędów w kompilatorze lub jego środowisku (sprzęt, system operacyjny), błędów projektowania w kompilowanym programie, niewłaściwego rozumienia języka źródłowego, błędów przepisywania, niepoprawnych danych itd.

Zadania procesu obsługi błędów
- wykrycie każdego błędu
- powiadomienie o nich użytkownika
- wykonanie (jeśli to tylko możliwe)napraw umożliwiających kontynuowanie przetwarzania.

Procedury obsługi błędów wykonuje się stosunkowo rzadko, w porównaniu z innymi zadaniami kompilatora, dlatego więc czas obsługi błędu jest w dużym stopniu nieistotny, jednak sam proces obsługi nie może być uważany za zbędny. Obsługa błędów ma olbrzymi wpływ na cały proces projektowania, a ponadto stanowi niezbędne narzędzie wspomagające samego kompilatora.

Generalnie możemy podzielić błędy na dwie grupy: błędy wykrywane podczas kompilacji oraz błędy wykrywane podczas wykonywania programu.

Błędy czy objawy ?
Nie można w sposób jednoznaczny określić przyczyny błędów, można jedynie diagnozować na podstawie widocznych objawów. Żadna naprawa nie może być traktowana jako *korekta* (w rozumieniu takim, jakim oczekiwałby tego programista). Naprawa neutralizuje tylko objawy , tak, aby proces mógł być kontynuowany.

Rozróżniamy rzeczywisty błąd i objawy. Metoda obsługi błędów, podobnie jak lekarz widzi, wyłącznie objawy. Na ich podstawie następuje próba postawienia diagnozy błędu. Diagnoza wiąże się zawsze z niepewnością, może więc lepiej wybrać zawiadomienie o objawach bez żadnych dalszych prób diagnozy. Zamiast słowa „błąd” bardziej odpowiednie byłoby słowo „objaw”.

Prostym przykładem rozróżnienia między objawem a błędem jest użycie niezadeklarowanego identyfikatora. Użycie jest tylko objawem, który mógł powstać w wyniku:
- popełnienia błędu w pisowni tego identyfikatora;
- popełnienia błędu w pisowni deklaracji lub pominięcia deklaracji;
- uszkodzenia struktury składniowej, powodującej wypadnięcie tego identyfikatora poza zasięg deklaracji;

Większość kompilatorów zawiadamia po prostu o objawach, pozostawiając diagnozowanie użytkownikowi.
Błąd jest wykrywany wtedy i tylko wtedy, gdy wynikiem jego jest ob. Jaw na ruszający definicję języka. Oznacza to w praktyce, że procedura obsługo błędów zależy od definicji języka, a nie zależy od analizowanego programu źródłowego.

Implementując proces obsługi błędów należy starać się, aby zawiadomienie o każdym z nich podano zostało jak najwcześniej. Jeśli objaw może być zauważony podczas kompilacji, należy o nim poinformować. Warto jednak zachować pewną ostrożność i nie zawiadamiać o błędach prze wystąpieniem ich objawów.
(Np. wyrażenie dzielenie przez zero jest zgodne zarówno ze składnią jak i z semantyką, natomiast objaw – dzielenie przez zero występuje tylko podczas wykonania programu czasie obliczenia wartości wyrażenia. Warto zatem aby kompilator nie zawiadamiał o błędzie w tym przypadku nawet gdyby go wykrył, w rzeczywistości bowiem to wyrażenie może być nigdy nie wyliczone i wówczas program będzie poprawny.)

Reakcje na objawy
Rozróżniamy trzy zasadnicze poziomy reakcji na objawy:
- **raport:** Przekazanie użytkownikowi sygnału o wystąpieniu błędu. Określenie objawu, wskazanie dokładnego miejsca jego wystąpienia i próba postawienia diagnozy.
- **wydobywanie z błędów:** Doprowadzenie procesu (kompilacji, wykonania) do logicznego stanu i kontynuacja w celu znalezienia dalszych błędów.
- **naprawa:** Próba postawienia diagnozy błędu na podstawie zaobserwowanego objawu. Jeśli diagnoza wydaje się poprawna, dokonanie stosownej zmiany w programie lub danych i kontynuacja przetwarzania.

Raport zawiera:
- numer błędu
- pozycję w tekście źródłowym
- podana w formie opisu istota wydobycia się kompilatora z błędu

Poziomy raportów

- o anomaliach
1) Uwaga
2) Komentarz – krytyka stylu programowania
3) Ostrzeżenie – odnoszą się do możliwych błędów

- o rzeczywistych błędach lub pogwałceniu ograniczeń
4) Błąd – mogą być naprawione
5) Błąd zgubny – wstrzymują tworzenie programu wykonywanego
6) Błąd śmiertelny – przerwanie kompilacji

Def: Błąd (pozycja, powaga, kod)

Błędy wykryte w czasie kompilacji są wykrywane podczas analizy programu źródłowego.

Def: Błędy zgubne
Nieekonomiczne lub niemożliwe do naprawy. Kompilator po osiągnięciu miejsca wystąpienia błędu wyprodukować kod przerywający ten program.

Def: Błędy śmiertelne
Pewne błędy (lub ograniczenia) uniemożliwiające kompilatorowi kontynuowanie działań.

Podstawowym wymaganiem, które muszą spełniać techniki wydobywania z błędów jest to, aby system nie przerywał działania w wyniku wystąpienia błędu, gdyż wówczas nie byłoby żadnego komunikatu o błędzie *(a nawet informacji o dokładnym położeniu objawu).*

Kompilator powinien wydobywać się z prawie wszystkich objawów, wykrywając tyle, ile tylko jest możliwe w pojedynczym wykonaniu. Istnieją jednak pewne błędy, które uniemożliwiają kompilatorowi wykonanie (tzw. błędy śmiertelne), w takiej sytuacji kompilator powinien poinformować o zaistniałej sytuacji i zakończyć działanie.

Proces wydobywania wymaga by kompilator dokonywał pewnej zmiany swojego stanu w celu osiągnięcia zgodności. Zmiana ta może spowodować pojawienie się rzekomych błędów w dalszym tekście. Który w rzeczywistości jest poprawny. Takie rzekomy błędy tworzą lawinę, a jednym z głównych kryteriów dla schematu wydobywania jest minimalizacji lawin.

Jeśli kompilator może diagnozować i naprawiać wszystkie błędy z dużym prawdopodobieństwem sukcesu, to program można bezpiecznie wykonać, pozwalając na wykrycie dalszych błędów. Naprawa wymaga jednak przezorności, ponieważ koszt wykonania może być bardzo wysoki, a szczególny charakter naprawy może spowodować bezużyteczność wykonania lub zniszczenie ważnych zbiorów danych. Najlepiej nie wykonywać poprawiania, chyba, że na wyraźne życzenie użytkownika.

Wszystkie błędy wykryte podczas kompilacji są wykrywane podczas analizy programu źródłowego. W czasie syntezy programu wykrywa się jedynie błędy kompilatora lub pogwałcenia ograniczeń. Są one zawsze zgubne.
Błędy wykryte podczas analizy można sklasyfikować ze względu na wykowane w danej chwili zadanie analizy:
- **leksykalne:** błędy w budowie jednostek, np. nielegalne znaki lub słowa kluczowe napisane z błędami;
- **składniowe:** błędy w budowie struktur; np. brakujące operatory lub nawiasy;
- **semantyczne:** błędy w zgodności, np. niezgodność typów argumentów z typem operatora lub niezadeklarowane zmienne.

Aby wydobywanie z błędów powiodło się całkowicie, każde z zadań analizy musi naprawiać wykryte błędy, a poprawny wynik przekazywać następnemu zdaniu. Niestety naprawa zwykle prowadzi raczej do lokalnego poprawienia błędów, umożliwia jednak wykrycie związanych z nim błędów przez następne zadania mające więcej informacji kontekstowej.

Każdy schemat wydobywania się z błędów musi bazować na nadmiarowej informacji pojawiającej się w programie. Im większa jest nadmierność tej informacji, tym łatwiejsze i bardziej pewne jest wydobywanie się. Ponieważ liczba struktur dostępnych procedurze wydobywania się z błędów istotnie wzrasta od poziomu leksykalnego do poziomu semantycznego, poprawne wydobywanie się z błędów semantycznych jest łatwiejsze niż wydobywanie się z błędów leksykalnych.

Strategie odzyskiwania kontroli (ang, error recovery)

- Tryb paniki

- Poziom frazy

- Produkcja dla błędów

- Korekta globalna

Opisy poszczególnych metod

Tryb paniki

Podczas analizy błąd jest wykrywany gdy terminal na wierzchołku stosu nie pasuje do kolejnego symbolu wejściowego, bądź gdy na wierzchołku stosu jest nieterminal A, kolejnym symbolem wejściowym jest b, a tablica analizatora nie zawiera żadnej wartości na pozycji M[A,b].
Odzyskiwanie kontroli w trybie paniki polega na pomijaniu symboli wejściowych aż do natrafienia na symbol leksykalny, który jest elementem wybranego zbiory symboli synchronizacyjnych. Efektywność tej metody zależy od wyboru zbioru synchronizacyjnego. Zbiory powinny być wybierane tak, aby analizator szybko obsługiwał błędy , które często występują w praktyce.

Jak znajdywać zbiory synchronizacyjne?

- W zbiorze synchronizacyjnym dla nieterminala A możemy umieścić wszystkie eleaty zbioru FOLLOW(A). Jeśli będziemy pomijać elementy aż do napotkania elementu z FOLLOW(A) i zdejmiemy A ze stosu to na pewno analiza będzie mogła być kontynuowana.

- Użycie samego zbioru FOLLOW(A) nie jest jednak wystarczające. Np. jeśli średniki kończą instrukcje, tak jak w języku C, to słowa kluczowe, które rozpoczynają instrukcje, mogą nie znaleźć się w zbiorze FOLLOW nieterminala tworzącego wyrażenia. Pominięcie średnika po przypisaniu może wówczas skutkować pominięciem słowa kluczowego rozpoczynającego kolejną instrukcję. Często istnieje hierarchiczna struktura konstrukcji języka, np. wyrażenia występują w instrukcjach, które są w blokach. Symbole rozpoczynające konstrukcje będące wyżej w hierarchii możemy dodać możemy dodać do zbioru synchronizacyjnego konstrukcji leżącej niżej. Przykładowo możemy dodać słowa kluczowe, które rozpoczynają instrukcje, do zbiorów synchronizacyjnych nieterminali tworzących wyrażenia.

- Jeśli dodamy zbiór FIRST(A) do zbioru synchronizacyjnego nieterminala A to umożliwimy wznowienie analizy zgodnie z A, jeśli symbol z FIRST(A) pojawi się na wejściu.
- Jeśli nieterminal może wygenerować ciąg pusty, to produkcja, z której wyprowadza się ε, może być użyta jako domyślna. Może to opóźnić wykrycie błędu, ale błąd na pewno zostanie zauważony. To podejście zmniejsza liczbę nie terminali, które trzeba rozważać podczas odzyskiwania kontroli.

- Jeśli terminal na wierzchołku stosu nie może być dopasowany, prostym sposobem jest zdjęcie tego terminala i wypisanie informacji mówiącej, że terminal został wstawiony

i kontynuowanie analizy. Takie podejście traktuje zbiory synchronizacyjne symbolu jako składające się ze wszystkich innych symboli.

Przykład.

Rozważmy gramatykę wyrażeń arytmetycznych:

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

Po eliminacji lewostronnej rekursji:

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid id$

Tablica analizatora dla tej gramatyki zawiera wartości „synch" oznaczające symbole synchronizacyjne pobrane ze zbioru FOLLOW rozpatrywanego nieterminala.

Nieterminal	Symbol wejściowy					
	Id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$	Synch	Synch
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$	Synch		$T \rightarrow FT'$	Synch	Synch
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow id$	Synch	Synch	$F \rightarrow (E)$	Synch	Synch

Jeśli analizator sprawdza wartość M[A,a] i stwierdza, że nie ma na niej wartości, to pomija symbol a.

Jeśli wartością jest synch, to nieterminal jest zdejmowany z wierzchołka stosu i podejmowana jest próba kontynuowania analizy.

Jeśli symbol na wierzchołku stosu nie pasuje do symbolu wejściowego, to zdejmujemy symbol ze stosu.

Spróbujmy zatem sparsować niepoprawne słowo: **) id * + id**

STOS	WEJŚCIE	WYJŚCIE
$E	**) id * + id$**	Błąd, pomiń (
$E	**id * + id$**	Id jest w FIRST(E)
$E'T	**id * + id$**	
$E'T'F	**id * + id$**	
$E'T'id	**id * + id$**	
$E'T'	*** + id$**	

$E'T'F*	*** + id$**	
$E'T'F	**+ id$**	Błąd, M[F,+] = synch
$E'T'	**+ id$**	F zostało zdjęte
$E'	**+ id$**	
$E'T+	**+ id$**	
$E'T	**id$**	
$E'T'F	**id$**	
$E'T'id	**id$**	
$E'T'	**$**	
$E'	**$**	
$	**$**	