반도체 공정 제어 시스템에서 변경된 코드의 영향 분석

강동옥, 박대준, 이영석, 이광근

프로그래밍 연구실
서울대학교

ROSAEC Workshop, 2013년 1월 30일 ~ 2월 2일

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 NRF ERC

목차

- 소개
- 목표
  - 분석 대상
- 분석
  - 큰 그림
  - 앞단
  - 뒷단
- 할일 및 결론

소개

- 반도체 산업 규모
  - 한 해(2012년) 매출 300억 달러
  - 세계 반도체 점유율 10%
  - 세계 2위
- 반도체 제조공정 관리시스템
  - 반도체와 역사를 같이 한 오래된 시스템
  - 총 인력 : 400여명
  - 순수 개발자 : 200여명

소개

- 반도체 제조공정 관리시스템
  - 제품 상태 추적
  - 스케쥴링
  - 데이터베이스 업데이트

- 소스코드가 수시로 변경된다.
  - 새로운 제조규칙이 생길 때마다 Rule Package 가 변경됨

소개

- 무결성의 중요성
  - 24시간 공장 가동 (효율의 극대화)
  - 잠시만 멈춰도 천문학적 손실

- 코드의 변경 시, 의도치 않은 부작용의 유무가 매우 중요함

소개

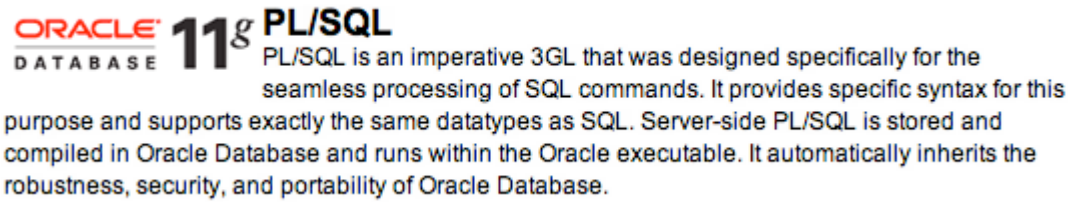
- 그러나 현실은
  - 품질 관리자 : 8명 (최근까지 2명)
  - 실제 사고 사례의 테스트 위주
    - 예기치 못한 문제는 찾기 어려움
  - 동료들의 검토 (Peer Review)
    - 코드 전부를 모두 이해하는 사람이 없음

목표

- 코드의 두 리비전 사이의 의미 차이 분석
  - 변경된 부분의 영향도 분석

- 분석 결과를 통해 의도치 않은 변경을 쉽게 발견 할 수 있게 함
  - 기존 Peer Review의 강화
  - 사고를 미리 예방하는 효과

분석 대상 (PL/SQL)



Oracle SQL +

절차형 언어 +

커서

분석 대상 (대상 코드)

- Rule Package
  - 여러 DB테이블을 참고하여 입력을 처리
  - 처리 결과를 적절한 DB에 업데이트

  - 어떤 '조건' 일 때, 어떤 '행동'을 하는지 기술
  - 핵심 연산은 문자열 생성 및 검색

분석 대상 (코드 변경)

- 작성되어 있는 코드나 변수의 삭제는 거의 없음
- 사용할 변수는 새로 만들어 씀
- 대체로 마이너 업그레이드
  - 사소한 로직 수정

분석 대상 (코드 변경의 예)

- 조건문의 변화

errMsg := …	IF b = 'N' THEN errMsg:= … ELSE …

IF a = 2 THEN
  x := t.attr1
ELSE
  x:= t.attr2
	IF a is null THEN
	  errMsg := …
	ELSIF a = 2 THEN
	  x := t.attr1
	ELSE
	  x:= t.attr2

분석 대상 (코드 변경의 예)

- DB query문 변화

INSERT INTO tt (tt_col1 ,tt_col2 ,tt_col3) VALUES (ee ,'ddd' ,aa.field1 || '-test')	INSERT INTO tt (tt_col1 ,tt_col2 ,tt_col3 ,tt_col4) VALUES (ee ,'ddd' ,aa.field1 || '-test' , 1)

방향

- 코드의 변경이 복잡하지 않음
  - 리팩토링과 같은 부분은 없음

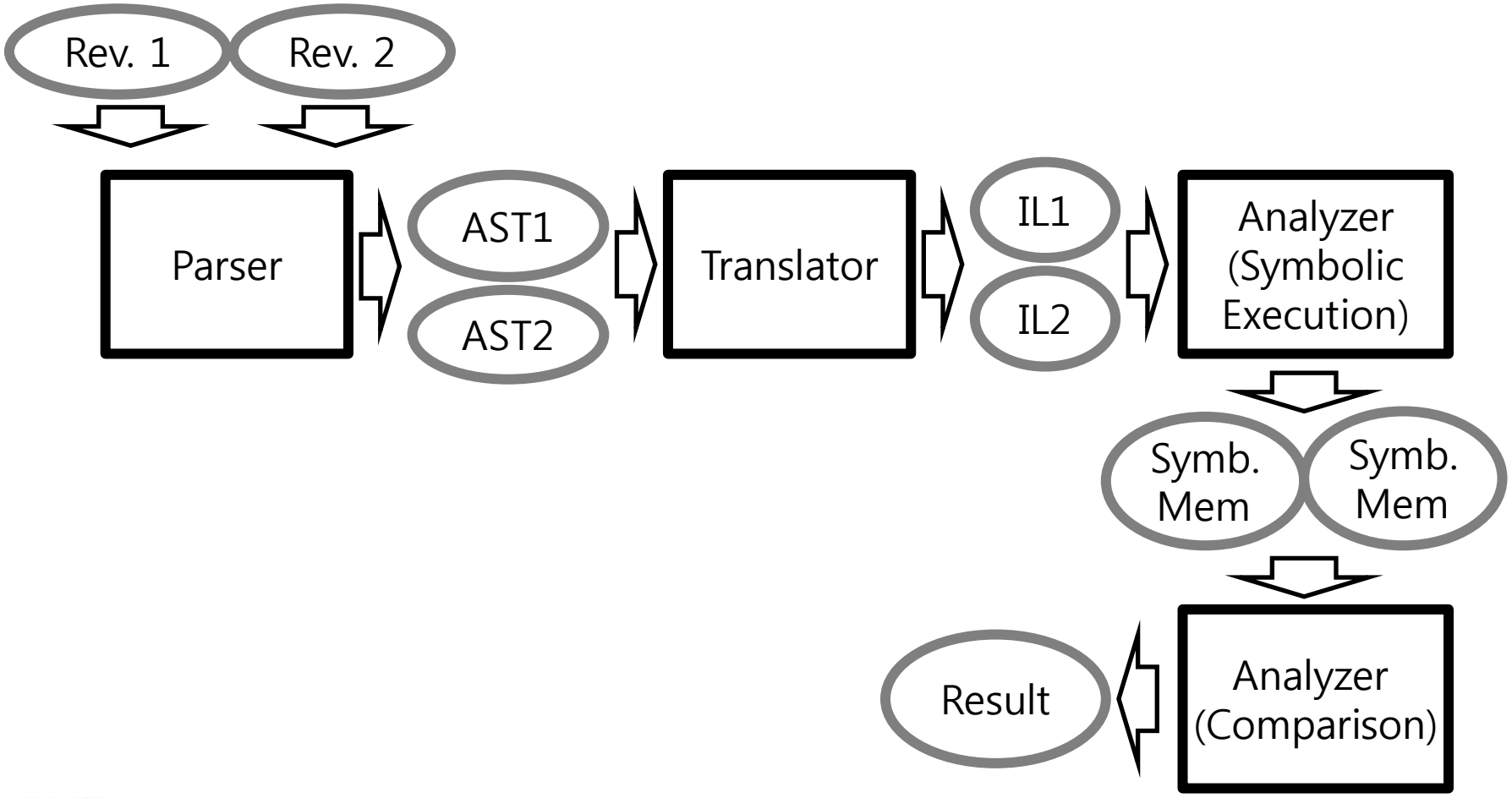
- 간단한 심볼릭 실행을 이용한 분석으로 시작함

목표 (자세히)

- 변경의 정의
  - 함수의 리턴 값, 함수 호출 시 인자들의 값, DB query에 사용된 변수의 변경

- 해당 값과 변수들이 왜 변경이 되었는가?
  - 영향도 분석

큰 그림

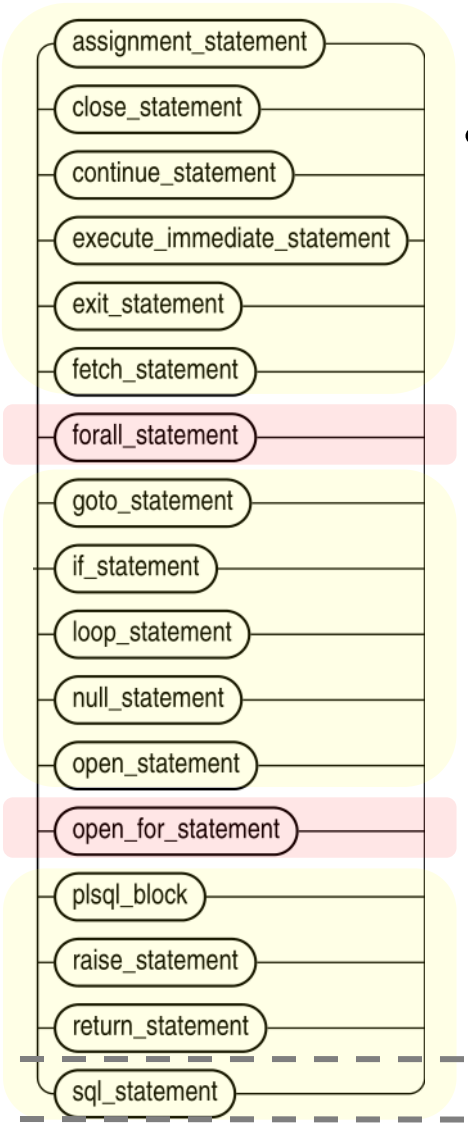


앞단 - 파서

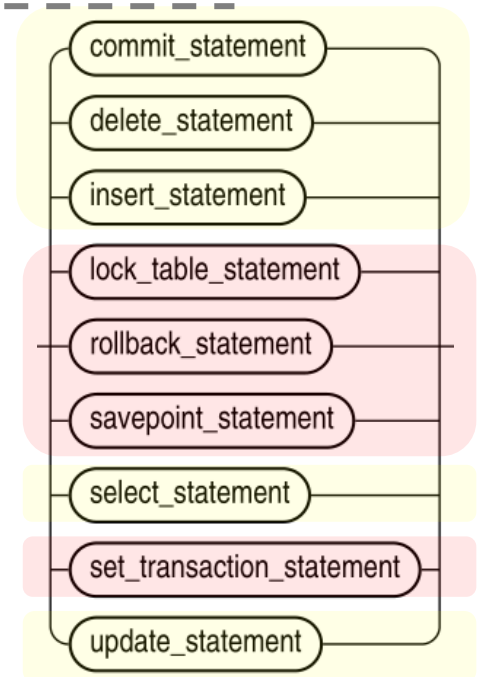
- 삼성에서 사용하는 PL/SQL 문법을 포괄하도록 구현

- PL/SQL 11g reference & SQL 11g reference 참조

앞단 - 파서



- 분석 대상 코드는 PL/SQL 문법의 일부분만 사용



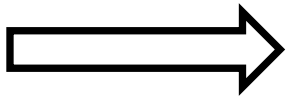
ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 NRF ERC

앞단 - 변환

- 파싱된 AST를 IL로 변환

package	→	*var external_decl*⋆	
external_decl	→	FUNCTION *type var* (*type var*)⋆ *block*	
	|	PROCEDURE *var* (*type var*)⋆ *block*	
block	→	*decl*⋆ *stmt*⋆ *exception_handler*⋆	
exception_handler	→	*var stmt*⋆	
	|	OTHERS *stmt*⋆	
decl	→	*type var exp*?	(w/ initialization)
	|	CURSOR *var* (*exp*⋆) *select*	(parameterized cursor)
	|	*typedef*	
typedef	→	TABLE *type type var*	
	|	RECORD (*type var exp*?)$^+$ *var*	(initialized fields)
	|	VARRAY *type var*	
type	→	*native_type*	
	|	*id*	(type name)
	|	TYPEOF *lexp*	
stmt	→	*label stmt_unit*	
stmt_unit	→	*lexp* := *exp*	
	|	IF *exp stmt*⋆ (ELSE *stmt*⋆)?	
	|	LOOP *stmt*⋆ | WHILE *exp stmt*⋆ | FOR *id exp stmt*⋆	
	|	GOTO *label* | EXIT *label*? *exp*? | CONTINUE *label*? *exp*?	
	|	*id* (*exp*⋆)	(proc call)
	|	RETURN *exp*?	
	|	*sql*	
	|	OPEN *var* (*exp*⋆)	
	|	CLOSE *var*	
	|	FETCH *var id*$^+$	(from, to)
	|	*block*	
sql	→	*select id*$^+$	(from, to)
	|	DELETE *table exp*	(cond)
	|	UPDATE *table* (*id exp*)$^+$ *exp*	((field, value)s, cond)
	|	INSERT *table* (*id*$^+$)? *exp*$^+$	(fields, values)
select	→	SELECT *table*$^+$ *id*$^+$ *exp*?	(fields, cond)
	|	*select sop select*	
sop	→	UNION | MINUS | INTERSECT | UNIONALL	
table	→	*id*	
	|	*select*	
exp	→	*lexp*	(incl. func call)
	|	*const* | *exp bop exp* | *uop exp* | (*exp*$^+$)	
id	→	*var*$^+$	
lexp	→	(*var* (*exp*⋆))⋆	(incl. collection ref)
var	→	*string*	
label	→	*string*	
string	→	⋯	
native_type	→	⋯	

Figure 1: PL/SQL AST Definition



Packages	*pkg*	→	*x proc*⋆ (*τ x*)⋆	(declarations)
Procedures	*proc*	→	*x stmt*	
Types	*τ*	→	INT | REAL | STRING	(primitive types)
		|	table *τ τ*	(key type, value type)
		|	array *τ e*	(value type, size)
		|	record (*τ x*)$^+$	
		|	fun *τ* (*τ x*)⋆	(return type, parameters)
Statements	*stmt*	→	*le* := *e*$^+$	(incl. record value)
		|	if *cond stmt stmt*	
		|	while *cond stmt*	
		|	goto *label*	
		|	label *label*	
		|	call *le e*⋆	(procedure call)
		|	return *e*	
		|	try *stmt* (*x stmt*)⋆	
		|	raise *x*	
		|	*stmt*; *stmt*	
Conditions	*cond*	→	*e rop e*$^+$	(incl. subset relation)
		|	*cond lop cond*	
		|	not *cond*	
L-expressions	*le*	→	*x*	
		|	*le.x*	
		|	*le*[*e*]	
Expressions	*e*	→	*le*	
		|	*const*	
		|	*e bop e*	
		|	*uop e*	
		|	*sql*	
		|	*le*(*e*⋆)	(function call)
Constants	*const*	∈	{NULL} ∪ ℤ ∪ ℝ ∪ *strings*	
SQLs	*sql*	→	SELECT *e*$^+$ *e*$^+$ *cond*	(table, fields, cond)
		|	INSERT *le e*$^+$	(table, record value)
		|	UPDATE *le* (*le e*)$^+$ *cond*	(table, (field, value)s, cond)
		|	DELETE *le cond*	(table, cond)
		|	*sql sop sql*	
Variables	*x*			
Labels	*label*			
BinaryOps	*bop*			
UnaryOps	*uop*			
RelationalOps	*rop*			
LogicalOps	*lop*			
SQLqueryOps	*sop*			

Figure 2: IL Definition

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 NRF ERC

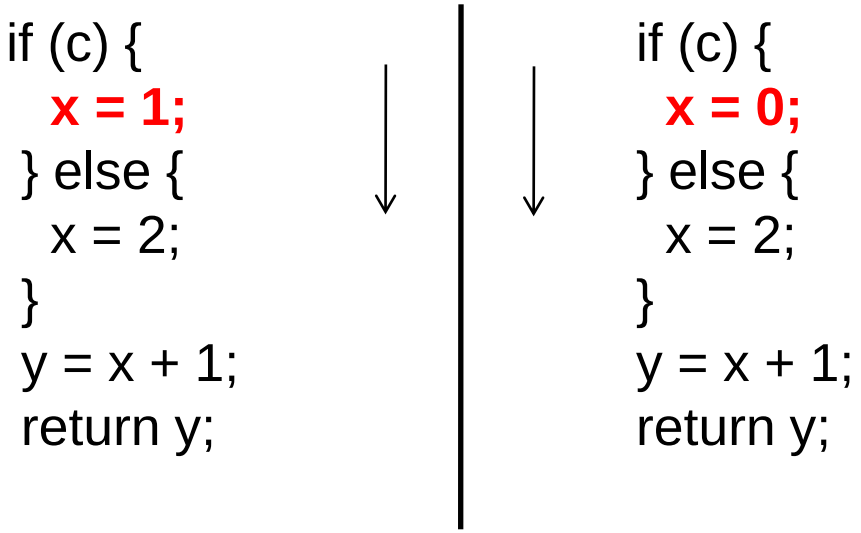
앞단 - 변환

x = f(1);

- 배열과 함수를 구별하기 위해 변환 시에 타입정보를 따로 저장 해야 함

- 나머지 부분의 변환은 간단한 맵핑

분석



SSA form 으로 바꾼 후에
심볼릭 실행

분석

```
if (c) {
  x = 1;
} else {
  x = 2;
}
y = x + 1;
return y;
```

x1: **1**, c

x2: 2, !c

x3: ϕ(x1,x2), true

y4: x3+1, true

```
if (c) {
  x = 0;
} else {
  x = 2;
}
y = x + 1;
return y;
```

x1: **0**, c

x2: 2, !c

x3: ϕ(x1,x2), true

y4: x3+1, true

분석

- 함수 호출의 경우

```
x = 1;
y = 2;
z = foo(x,y);
return z;
```

```
x1 = 1;
y2 = 2;
z3 = foo(x1);
return z3;
```

```
foo(a,b) {
print b;
return a + 1;
}
```

분석

- 각각의 함수를 바닥부터 SSA form으로 변환
- 리턴 값에 영향을 주는 인자들 분석
- 함수 단위로 비교

할일 및 결론

- 삼성에서 제공한 테스트 케이스에 대해 분석이 잘 되는 것을 확인
- 회사와의 피드백을 통해 분석기를 보완

- Peer Review의 보완 혹은 대체수단이 될 수 있음
- 간단한 정적 분석으로 실제 산업에 유용하게 쓰일 수 있는 분석기 구현

감사합니다.

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 NRF ERC