

# TRIPS to the Semantic EDGE

Ales Kozumplik   Martin Krivanek   Antoine Madet  
Original project proposed by Rene Rydhof Hansen

Aalborg University, Denmark

MEMICS 2008, Znojmo, Czech Republic

Formal semantics for programming languages, why?

- Overcome complexity / ambiguities
- Correctness of program analysis
- Code equivalence verification
- Model-checking...

TRIPS, a novel microprocessor architecture based on EDGE

Explicit Data Graph Execution (EDGE) instruction set architecture

- high-performance
- scalability
- instruction-level parallelism

**Complex** processing model

⇒ Need for a clear semantics

- Small-step semantics of a representative subset of TASL instructions (TRIPS assembly language)
- Insights into termination and non-determinism aspects
- Simulator for validation purposes

- 1 TRIPS Overview
- 2 Small-Step Semantics
- 3 Properties
- 4 Conclusion

- 1 TRIPS Overview
- 2 Small-Step Semantics
- 3 Properties
- 4 Conclusion

# The TRIPS processing model: 2 levels

## Block-level ( $\leq 128$ instructions)

- program split into **sequentially** executed blocks
- blocks are **atomic**
- successful execution  $\Rightarrow$  **block commit step**

## Instruction-level (inside blocks)

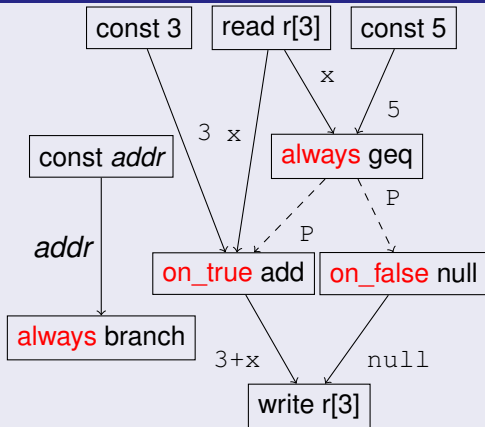
- direct instruction communication
- instructions execute in **data-flow order**
- control flow is done through **predicated instructions**
- exception propagation

# A block example

## C code

```
if (x >= 5)
    x = 3 + x;
```

## Data-flow graph



- 1 TRIPS Overview
- 2 Small-Step Semantics**
- 3 Properties
- 4 Conclusion

# Instruction-level semantics

## Preliminary definitions

- Block:  $B \in \mathcal{P}(Instruction)$
- Block context:  $\mathcal{C} \in Context$   
 $Context = Target + Status \rightarrow$   
 $Integer + Bool + Exception + Nullification$
- **Local** state:  $\sigma \in State = Register \rightarrow Integer$
- **Local** memory:  $\mu \in Memory = Address \rightarrow Integer$

## Instruction-level transition relation

$$\langle B, \mathcal{C}, \sigma, \mu \rangle \Rightarrow_i \langle B', \mathcal{C}', \sigma', \mu' \rangle$$

# Inference rules – a typical case

## ADD instruction

$$\text{ADD} \frac{\iota = (n[n] P \text{ add } T_1 T_2) \in B}{\langle B, C, \sigma, \mu \rangle \Rightarrow_i \langle B \setminus \{\iota\}, C[T_1 \mapsto v, T_2 \mapsto v], \sigma, \mu \rangle}$$

if  $\{ \phi_{\text{readyOperands}} \wedge \phi_{\text{matchingPredicate}}$

where  $v = \begin{cases} C(n[n, 0]) + C(n[n, 1]) & \text{if } \neg\phi_{\text{exception}} \wedge \neg\phi_{\text{null}} \\ \text{NULL} & \text{if } \neg\phi_{\text{exception}} \wedge \phi_{\text{null}} \\ \text{EXCEPT} & \text{if } \phi_{\text{exception}} \end{cases}$

## Exceptions

Any output instruction (`write/store`) receive exception

⇒ ensure *block commit step* **failure**

How? Raise an exception flag:  $C(\mathcal{E}) = \text{True}$

## Load dependences

`load` instructions present a special challenge

⇒ use statically assigned **priority numbers** to defer execution after all `stores` they depend on

# Block-level semantics

## Preserving atomicity

$commitReady : Block \times Context \rightarrow Bool$

## Block-level transition relation

$\langle s, m \rangle \Rightarrow_b \gamma$  where  $\gamma = \langle s', m' \rangle$  or  $\gamma = \perp$

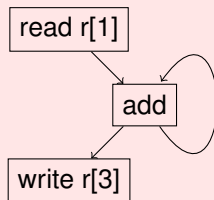
## Successful block commit step

$$\text{BLOCK} \frac{\begin{array}{l} B_{start} = BQ[s(PC)] \\ \sigma = s \quad s' = \sigma' \quad \mu = m \quad m' = \mu' \\ \langle B_{start}, C_{start}, \sigma, \mu \rangle \Rightarrow_i^+ \langle B', C', \sigma', \mu' \rangle \end{array}}{\langle s, m \rangle \Rightarrow_b \langle s', m' \rangle}$$

$$\text{if } \left\{ \begin{array}{l} commitReady(B_{start}, C') \\ C'(\mathcal{E}) \neq True \end{array} \right.$$

- 1 TRIPS Overview
- 2 Small-Step Semantics
- 3 Properties**
- 4 Conclusion

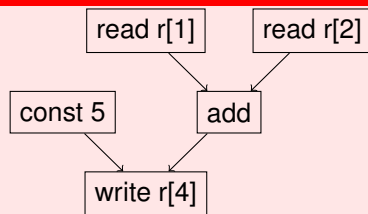
Does this block terminate?



## Property

A correctly formed block, which satisfies the conditions enumerated by *commitReady*, always terminates. Either successfully or by throwing an exception.

Is this block deterministic?



## Property

For all blocks where every operand, predicate, memory location and register is targeted exactly once and whose execution terminates, the block execution is deterministic. Formally:

$$\langle s, m \rangle \Rightarrow_b \gamma \wedge \langle s, m \rangle \Rightarrow_b \gamma' \implies \gamma = \gamma'$$

- 1 TRIPS Overview
- 2 Small-Step Semantics
- 3 Properties
- 4 Conclusion**

## Summary

- Small-step semantics of selected TRIPS instructions
- Conditions guaranteeing block termination and determinism
- Simulator of our semantics

## Future work

- Exception handling
- Function calls
- Full set of instructions. . .

Thank you. Questions?