A Brief Tour of Term Rewriting

Zack Grannan, CPSC 501

December 7th, 2021

Term Rewriting Systems

- **What are Term Rewriting Systems?**
  - A model for transforming *terms* via application of *rewrite rules*
  - Rules define structural transformations
- **Why are Term Rewriting Systems Useful?**
  - Can serve as a nondeterministic model of computation
  - Expressive, but with very simple syntax and semantics
  - Admit interesting properties (today: Termination and Confluence)

Today

1. Introduce Term Rewriting Systems
2. Describe Properties of Term Rewriting Systems
   1. Confluence
   2. Termination
3. Prove Termination of Term Rewriting Systems

Today

1. Introduce Term Rewriting Systems
2. Describe Properties of Term Rewriting Systems
    1. Confluence
    2. Termination
3. Prove Termination of Term Rewriting Systems

Term Rewriting Systems, Formally

From (Klop, 1990)

We consider terms built from an alphabet $\Sigma = (F, V)$ such that:

- F is a set of *function symbols*, each associated with an arity
  - The arity of a function is the number of arguments it is supposed to have
  - We denote the arity of a function f as $\alpha(f)$
- V is a countably infinite set of *variables* (typically denoted $x, x_1, y, y_1, \ldots$)

Term Rewriting Systems, Formally

From (Klop, 1990)

Terms over Σ are denoted as $T(\Sigma)$, where:

1. $x \in V \implies x \in T(\Sigma)$
2. $f \in F \land \alpha(f) = n \land t_1, \ldots, t_n \in T(\Sigma) \implies f(t_1, \ldots, t_n) \in T(\Sigma)$

To make things look nice:

- If $\alpha(f) = 0$, we can write $f()$ as f
- If $\alpha(f) = 2$, we can write $f(t, u)$ as $t\ f\ u$

Example: Addition and Peano Numbers

$F = \{z, s, + \}$

Example: Addition and Peano Numbers

$F = \{z, s, +\}$

z

Example: Addition and Peano Numbers

$F = \{z, s, + \}$

z

$s(z)$

Example: Addition and Peano Numbers

$F = \{z, s, +\}$

z

$s(z)$

$s(s(z))$

Example: Addition and Peano Numbers

$F = \{z, s, +\}$

z

$s(z)$

$s(s(z))$

$\ldots$

Example: Addition and Peano Numbers

$F = \{z, s, +\}$ $z + z$

z

$s(z)$

$s(s(z))$

$\ldots$

Example: Addition and Peano Numbers

$F = \{z, s, +\}$

z

$s(z)$

$s(s(z))$

$\ldots$

$z + z$

$z + s(z)$

Example: Addition and Peano Numbers

$F = \{z, s, +\}$

z

$s(z)$

$s(s(z))$

$\ldots$

$z + z$

$z + s(z)$

$s(z) + s(z)$

Example: Addition and Peano Numbers

$F = \{z, s, +\}$

z

$s(z)$

$s(s(z))$

$\ldots$

$z + z$

$z + s(z)$

$s(z) + s(z)$

$(z + z) + s(z)$

Example: Addition and Peano Numbers

$F = \{z, s, +\}$	$z + z$
z	$z + s(z)$
$s(z)$	$s(z) + s(z)$
$s(s(z))$	$(z + z) + s(z)$
$\ldots$	$\ldots$

Term Rewriting Systems, Informally

From (Klop, 1990)

A *rewrite rule* r is a pair of terms (t, u), denoted $r : t \to u$ or simply $t \to u$

A rewrite rule $r : t \to u$ defines a binary relation $\to_r$ on T, informally:

- $t' \to_r u'$ if t "matches" some subterm s of t' via a substitution σ
- u' is the result of replacing s in t' with substitution σ applied to u'

Term Rewriting Systems, Informally

From (Klop, 1990)

A *rewrite rule* r is a pair of terms (t, u), denoted $r : t \to u$ or simply $t \to u$

A rewrite rule $r : t \to u$ defines a binary relation $\to_r$ on T, informally:

- $t' \to_r u'$ if t "matches" some subterm s of t' via a substitution σ
- u' is the result of replacing s in t' with substitution σ applied to u'

Example:

- Rewrite Rule $r : x + z \to x$

Term Rewriting Systems, Informally

From (Klop, 1990)

A *rewrite rule* r is a pair of terms (t, u), denoted $r : t \to u$ or simply $t \to u$

A rewrite rule $r : t \to u$ defines a binary relation $\to_r$ on T, informally:

- $t' \to_r u'$ if t "matches" some subterm s of t' via a substitution σ
- u' is the result of replacing s in t' with substitution σ applied to u'

Example:

- Rewrite Rule $r : x + z \to x$
- Application $(s(z) + z)+s(z)$

Term Rewriting Systems, Informally

From (Klop, 1990)

A *rewrite rule* r is a pair of terms (t, u), denoted $r : t \to u$ or simply $t \to u$

A rewrite rule $r : t \to u$ defines a binary relation $\to_r$ on T, informally:

- $t' \to_r u'$ if t "matches" some subterm s of t' via a substitution σ
- u' is the result of replacing s in t' with substitution σ applied to u'

Example:

- Rewrite Rule $r : x + z \to x$
- Application $(s(z) + z) + s(z)$

Term Rewriting Systems, Informally

From (Klop, 1990)

A *rewrite rule* r is a pair of terms (t, u), denoted $r : t \rightarrow u$ or simply $t \rightarrow u$

A rewrite rule $r : t \rightarrow u$ defines a binary relation $\rightarrow_r$ on T, informally:

- $t' \rightarrow_r u'$ if t "matches" some subterm s of t' via a substitution σ
- u' is the result of replacing s in t' with substitution σ applied to u'

Example:

- Rewrite Rule $r : x + z \rightarrow x$
- Application $(s(z) + z) + s(z) \rightarrow_r s(z) + s(z)$

Term Rewriting Systems, Formally

From (Klop, 1990)

A *rewrite rule* r is a pair of terms (t, u), denoted $r : t \to u$ or simply $t \to u$

A *substitution* σ is a mapping $V \to T$

- $\sigma(t)$ denotes the replacement of each variable $v \in t$ with $\sigma(v)$

A *context* C is a term with a single "hole", e.g. $f(x, \square)$ or the trivial context $\square$

- $C[t]$ represents the term obtained by filling the hole with t, e.g $f(x, t)$ or t

A rewrite rule $r : t \to u$ defines a binary relation $\to_r$ where:

$C[\sigma(t)] \to_r C[\sigma(u)]$ for all contexts C, substitutions σ

Example

Klop

Rules:

$$r_1 : x + z \to x$$

$$r_2 : x + s(y) \to s(x + y)$$

Rewriting:

$$s(z) + s(s(z))$$

Example

Klop

Rules:

$r_1 : x + z \to x$

$r_2 : x + s(y) \to s(x + y)$

$\sigma = \{x \to s(z), y \to s(z)\}$

$C = \square$

Rewriting:

$s(z) + s(s(z)) \to_{r_2}$

$s(s(z) + s(z))$

Example

Klop

Rules:

$r_1 : x + z \to x$

$r_2 : x + s(y) \to s(x + y)$

$\sigma = \{x \to s(z), y \to z\}$

$C = s(\square)$

Rewriting:

$s(z) + s(s(z)) \to_{r_2}$

$s(s(z) + s(z)) \to_{r_2}$

$s(s(s(z) + z)))$

Example

Klop

Rules:

$r_1 : x + z \to x$

$r_2 : x + s(y) \to s(x + y)$

$\sigma = \{x \to s(z)\}$

$C = s(s(\square))$

Rewriting:

$s(z) + s(s(z)) \to_{r_2}$

$s(s(z) + s(z)) \to_{r_2}$

$s(s(s(z) + z)) \to_{r_1}$

$s(s(s(z)))$

Term Rewriting Systems, Finally

From (Klop, 1990)

A Term Rewriting System is the pair (Σ, R) where:

- Σ is an alphabet, and
- R is a set of rewrite rules over $T(\Sigma)$

Typically, the rewrite rules denote axioms for some theory

A TRS defines a binary relation $\rightarrow_R$ on $T(\Sigma)$, such that:

- $t \rightarrow_R u$ iff u can be obtained from t by applying a rule from R

Reasoning Enabled by TRS

A relation of interest is $\rightarrow_R^*$, the reflexive transitive closure of $\rightarrow_R$

Intuition: $t \rightarrow_R^* u$ iff u can be obtained from t by applying zero or more rewrite rules from R

A common application: compute the set of terms obtained via rewriting from t, namely the set: $\{u \in T \mid t \rightarrow_R^* u\}$

Reasoning Enabled by TRS

A relation of interest is $\rightarrow_R^*$, the reflexive transitive closure of $\rightarrow_R$

Intuition: $t \rightarrow_R^* u$ iff u can be obtained from t by applying zero or more rewrite rules from R

A common application: compute the set of terms obtained via rewriting from t, namely the set: $\{u \in T \mid t \rightarrow_R^* u\}$

Also of interest: obtaining the *normal form* of t.

Find a u such that $t \rightarrow_R^* u \wedge \neg(\exists v \,.\, u \rightarrow_R v)$

Example 2

$r_1 : T \wedge x \to x$

$r_2 : F \wedge x \to F$

$r_3 : T \vee x \to T$

$r_4 : F \vee x \to x$

$r_5 : \text{not}(T) \to F$

$r_6 : \text{not}(F) \to T$

$(T \vee F) \wedge \text{not}(F)$

Example 2

$r_1 : T \wedge x \rightarrow x$

$r_2 : F \wedge x \rightarrow F$

$r_3 : T \vee x \rightarrow T$

$r_4 : F \vee x \rightarrow x$

$r_5 : \text{not}(T) \rightarrow F$

$r_6 : \text{not}(F) \rightarrow T$

$(T \vee F) \wedge \text{not}(F)$

Example 2

$r_1 : T \wedge x \rightarrow x$

$r_2 : F \wedge x \rightarrow F$

$r_3 : T \vee x \rightarrow T$

$r_4 : F \vee x \rightarrow x$

$r_5 : \text{not}(T) \rightarrow F$

$r_6 : \text{not}(F) \rightarrow T$

$(T \vee F) \wedge \text{not}(F)$

$T \wedge \text{not}(F)$

$(T \vee F) \wedge T$

Example 2

$r_1 : T \wedge x \rightarrow x$

$r_2 : F \wedge x \rightarrow F$

$r_3 : T \vee x \rightarrow T$

$r_4 : F \vee x \rightarrow x$

$r_5 : \text{not}(T) \rightarrow F$

$r_6 : \text{not}(F) \rightarrow T$

$(T \vee F) \wedge \text{not}(F)$

$T \wedge \text{not}(F)$

$(T \vee F) \wedge T$

Example 2

$r_1 : T \wedge x \to x$

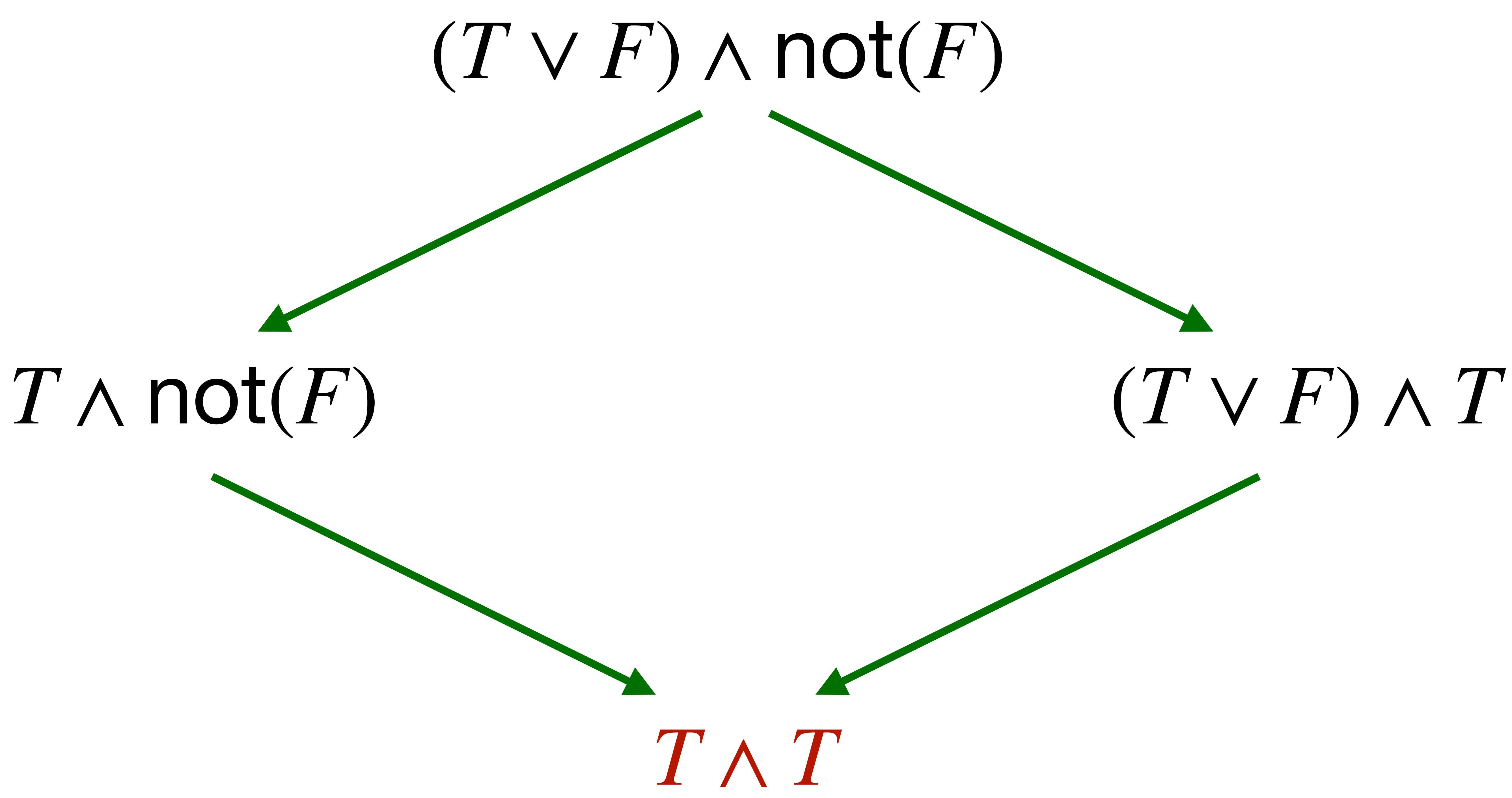
$r_2 : F \wedge x \to F$

$r_3 : T \vee x \to T$

$r_4 : F \vee x \to x$

$r_5 : \mathrm{not}(T) \to F$

$r_6 : \mathrm{not}(F) \to T$



Example 2

$r_1 : T \wedge x \to x$

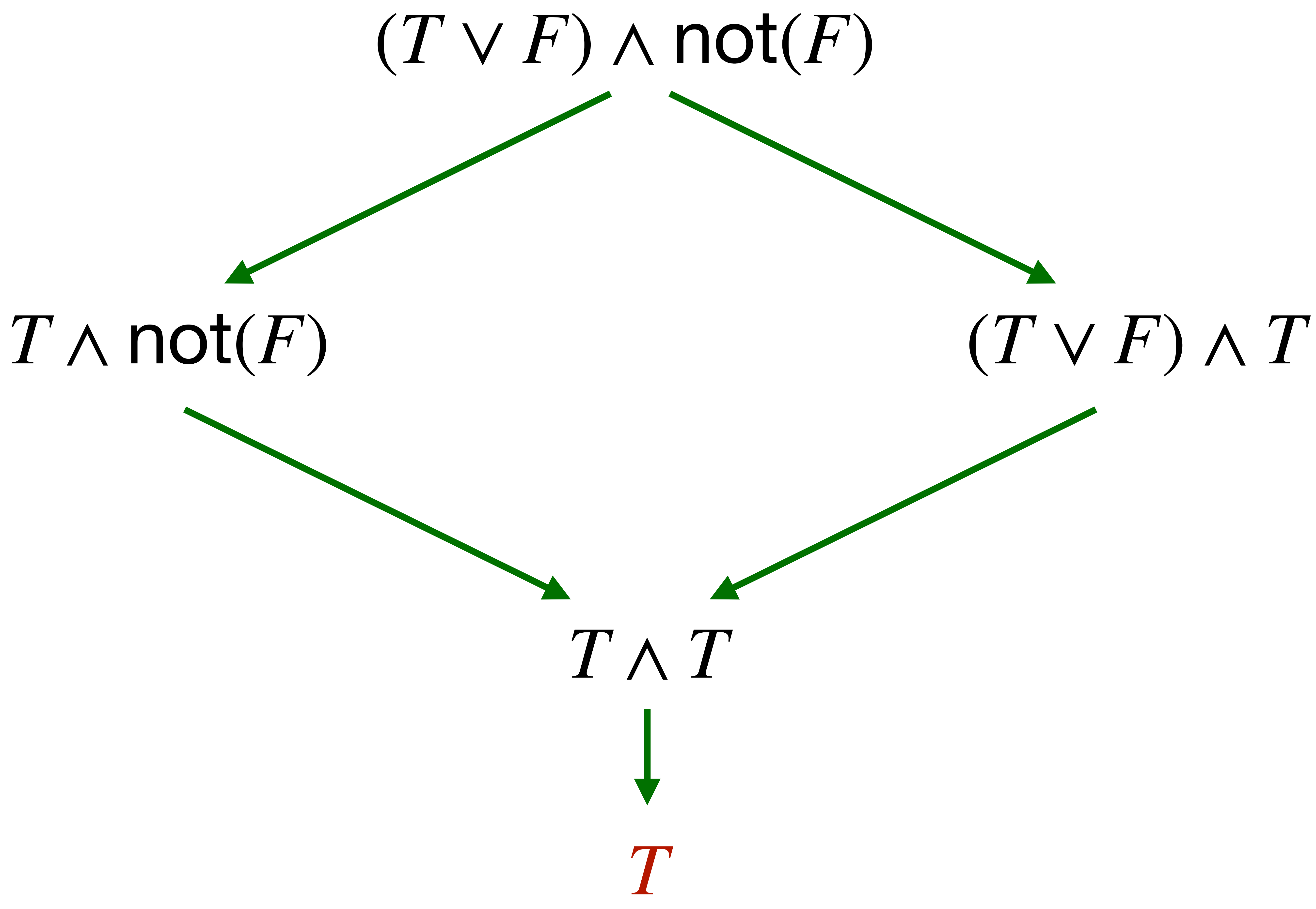
$r_2 : F \wedge x \to F$

$r_3 : T \vee x \to T$

$r_4 : F \vee x \to x$

$r_5 : \text{not}(T) \to F$

$r_6 : \text{not}(F) \to T$



Real-World Applications of Term Rewriting

- Automated Theorem Proving (Equality Saturation)
  - Egg (Willsey et al, 2021)
- Proving Program Termination
  - AProVe (Giesl, Thiemann, Schneider-Kamp, & Falke, 2004)
- Implementing Decision Procedures for Equational Theories
  - Knuth-Bendix Completion (Knuth & Bendix, 1983)

Today

1. Introduce Term Rewriting Systems
2. Describe Properties of Term Rewriting Systems
    1. Confluence
    2. Termination
3. Prove Termination of Term Rewriting Systems

Confluence

Informally, a TRS is *confluent* if the ordering of the rewrite steps do not matter.

Term rewriting is nondeterministic:

- A single rewrite rule might apply at different locations in a term
- Two rewrite rules may apply to the same term

Formally:

$$t \to_R^* u_1 \land t \to_R^* u_2 \implies \exists w \,.\, u_1 \to_R^* w \land u_2 \to_R^* w$$

Confluence

$$s(z + z) + z$$

Confluence

$$s(z + z) + z$$

$C = s(\square) + z$

$$s(z) + z$$

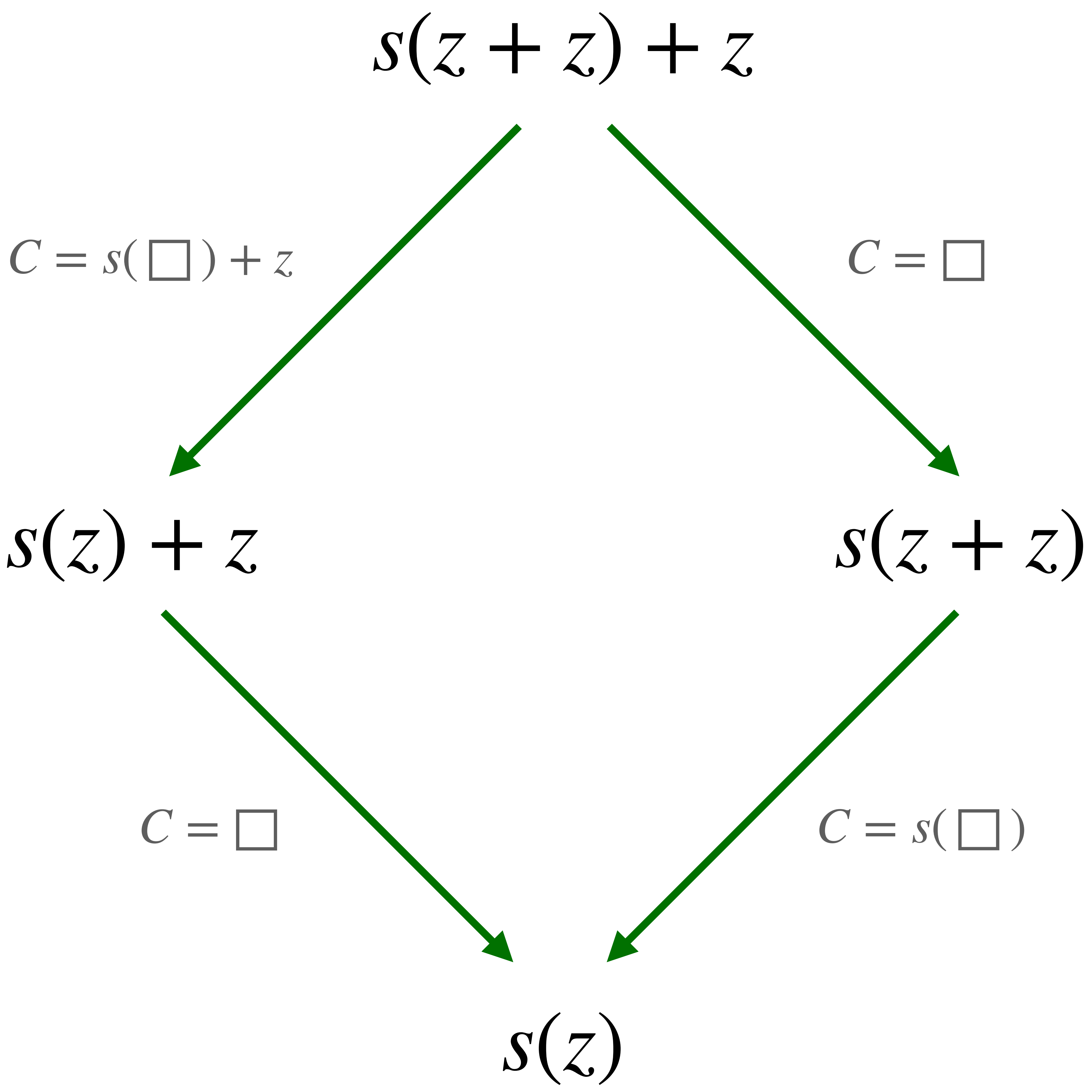
Confluence

$$s(z + z) + z$$

$C = s(\square) + z$ 		 $C = \square$

$$s(z) + z \qquad\qquad s(z + z)$$

Confluence



Termination

A TRS (Σ, R) is terminating if there do not exist infinite chains of the form:

$$t_1 \rightarrow_R t_2 \rightarrow_R \cdots$$

Termination

A TRS (Σ, R) is terminating if there do not exist infinite chains of the form:

$t_1 \rightarrow_R t_2 \rightarrow_R \cdots$

Consider a rule for "commutativity" $r : x + y \rightarrow y + x$

We could then have:

$s(z) + z$

Termination

A TRS (Σ, R) is terminating if there do not exist infinite chains of the form:

$t_1 \rightarrow_R t_2 \rightarrow_R \cdots$

Consider a rule for "commutativity" $r : x + y \rightarrow y + x$

We could then have:

$s(z) + z \rightarrow_r z + s(z)$

Termination

A TRS (Σ, R) is terminating if there do not exist infinite chains of the form:

$t_1 \rightarrow_R t_2 \rightarrow_R \cdots$

Consider a rule for "commutativity" $r : x + y \rightarrow y + x$

We could then have:

$s(z) + z \rightarrow_r z + s(z) \rightarrow_r s(z) + z$

Termination

A TRS (Σ, R) is terminating if there do not exist infinite chains of the form:

$t_1 \rightarrow_R t_2 \rightarrow_R \cdots$

Consider a rule for "commutativity" $r : x + y \rightarrow y + x$

We could then have:

$s(z) + z \rightarrow_r z + s(z) \rightarrow_r s(z) + z \rightarrow_r \ldots$

Why do we care?

- If a TRS for a given theory is terminating and confluent, then it defines a decision procedure for equality in that theory
  - Simply obtain the normal form for each term and compare
  - For more info see (Knuth & Bendix, 1983)
- Various techniques exist for proving termination of rewriting
  - If you want to prove your program terminates, express it as a TRS and prove termination of the TRS

Today

1. Introduce Term Rewriting Systems
2. Describe Properties of Term Rewriting Systems
   1. Confluence
   2. Termination
3. Prove Termination of Term Rewriting Systems

Termination

A TRS (Σ, R) is terminating if there do not exist infinite chains of the form:

$t_1 \rightarrow_R t_2 \rightarrow_R \cdots$

Is it possible to decide in general?

Termination

A TRS (Σ, R) is terminating if there do not exist infinite chains of the form:

$$t_1 \to_R t_2 \to_R \cdots$$

Is it possible to decide in general?

No! You can implement a Turing machine as a TRS!

In fact, a Turing machine can be implemented as a single rewrite rule.

Termination

A TRS (Σ, R) is terminating if there do not exist infinite chains of the form:

$t_1 \to_R t_2 \to_R \cdots$

Is it possible to decide sometimes?

A Simple Recipe for Proving Termination

1. Define a function $S : T \rightarrow \mathbb{N}$

A Simple Recipe for Proving Termination

1. Define a function $S : T \to \mathbb{N}$
2. Ensure that forall t, u we have $t \to_R u \implies S(t) > S(u)$

A Simple Recipe for Proving Termination

1. Define a function $S : T \to \mathbb{N}$
2. Ensure that forall t, u we have $t \to_R u \implies S(t) > S(u)$

Why this works:

For any initial term t, then $S(t)$ is an arbitrary integer n.

There are $n - 1$ numbers that are less than n, so the longest path starting at t has at most n steps.

Termination

Consider the following rewrite rules:

$r_1 : T \wedge x \to x$

$r_2 : F \wedge x \to F$

$r_3 : T \vee x \to T$

$r_4 : F \vee x \to x$

$r_5 : \text{not}(T) \to F$

$r_6 : \text{not}(F) \to T$

Termination

Consider the following rewrite rules:

$r_1 : T \wedge x \to x$

$r_2 : F \wedge x \to F$

$r_3 : T \vee x \to T$

$r_4 : F \vee x \to x$

$r_5 :$ not$(T) \to F$

$r_6 :$ not$(F) \to T$

Let $S(t)$ denote the number of symbols in the term.

For each rule r_i, we require $t \to_{r_i} u \implies S(t) > S(u)$

Termination

Consider the following rewrite rules:

$r_1 : T \wedge x \to x$

$r_2 : F \wedge x \to F$

$r_3 : T \vee x \to T$

$r_4 : F \vee x \to x$

$r_5 : \text{not}(T) \to F$

$r_6 : \text{not}(F) \to T$

Let $S(t)$ denote the number of symbols in the term.

For each rule r_i, we require $t \to_{r_i} u \implies S(t) > S(u)$

$S(t) > S(t) - 2$

$S(t) > S(t) - S(\sigma(x)) - 1$

$S(t) > S(t) - S(\sigma(x)) - 1$

$S(t) > S(t) - 2$

$S(t) > S(t) - 1$

$S(t) > S(t) - 1$

A General Recipe for Proving Termination

- Recall: The simple recipe worked by mapping elements of T to elements of $\mathbb{N}$
- An infinite descent of $\rightarrow_R$ on T would correspond to an infinite descent of $>$ on $\mathbb{N}$
- Infinite descent of $>$ on $\mathbb{N}$ is not allowed because $>$ is *well-founded* on $\mathbb{N}$.
-

A General Recipe for Proving Termination

- Recall: The simple recipe worked by mapping elements of T to elements of $\mathbb{N}$
- An infinite descent of $\rightarrow_R$ on T would correspond to an infinite descent of $>$ on $\mathbb{N}$
- Infinite descent of $>$ on $\mathbb{N}$ is not allowed because $>$ is *well-founded* on $\mathbb{N}$.
- Thus the general recipe for proving termination is:
    - Show that an infinite computation would correspond to an infinite descent in a well-founded relation

Conclusion

- Term Rewriting Systems can serve as expressive yet simple model of computation
- TRS admit interesting properties such as confluence and termination
- In some cases it is possible to prove termination of a TRS

Bibliography

Giesl, J., et al. (2004). Automated termination proofs with AProVE. International Conference on Rewriting Techniques and Applications, Springer.

Klop, J. W. and J. Klop (1990). Term rewriting systems, Centrum voor Wiskunde en Informatica.

Knuth, D. E. and P. B. Bendix (1983). Simple word problems in universal algebras. Automation of Reasoning, Springer: 342-376.

Willsey, M., et al. (2021). "Egg: Fast and extensible equality saturation." Proceedings of the ACM on Programming Languages (POPL): 1-29.